21世纪高等学校规划教材 | 软件工程





软件测试的概念与方法

聂长海 编著

清华大学出版社

21 世纪高等学校规划教材·软件工程

软件测试的概念与方法

聂长海　编著

清华大学出版社
北 京

内 容 简 介

本书以软件测试方法的分类为线索，以软件测试的各种方法为内容，系统地介绍各种软件测试方法的概念、理论、特点和在工程实践中应用的例子，内容力求简洁、清楚。与已有的软件测试教材和相关书籍相比，本书的特色是以各种软件测试方法为中心，系统地介绍到目前为止，几乎所有的可以查到或见到的软件测试方法，是对目前已经出版的软件测试书籍的一个重要补充。

全书共分 5 章，第 1 章为概论部分，介绍软件测试相关的概念、方法、历史发展、职业和学术研究等方面，给读者一个软件测试比较完整的视图，同时强调本书试图系统介绍软件的概念、理论和方法；第 2～6 章都是介绍各种软件测试方法，其中第 2 章介绍一般的白盒和黑盒测试方法，第 3 章介绍软件开发过程中各个阶段的软件测试，第 4 章介绍 31 种针对软件各种特性和各个方面的测试，第 5 章介绍 30 种专门的软件测试方法，第 6 章介绍 12 种利用新方法、新平台或者满足新要求的软件测试方法。全书每章各个小节后均附有习题。

本书不仅适用于高等院校计算机、软件工程专业高年级本科生、研究生作为教材使用，也适合作为计算机软件的开发人员、软件测试的从业人员、广大科技工作者和研究人员的参考书。

图书在版编目（CIP）数据

软件测试的概念与方法/聂长海编著. —北京：清华大学出版社，2013.5
21 世纪高等学校规划教材·软件工程
ISBN 978-7-302-31646-6

Ⅰ. ①软… Ⅱ. ①聂… Ⅲ. ①软件－测试－高等学校－教材 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 040707 号

责任编辑： 索　梅
封面设计：
责任校对： 李建庄
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 13.5　　**字　　数：** 339 千字
版　　次： 2013 年 5 月第 1 版　　**印　　次：** 2013 年 5 月第1 次印刷
印　　数： 1～　000
定　　价： .00 元

产品编号：041804-01

出版说明

随着我国改革开放的进一步深化,高等教育也得到了快速发展,各地高校紧密结合地方经济建设发展需要,科学运用市场调节机制,加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度,通过教育改革合理调整和配置了教育资源,优化了传统学科专业,积极为地方经济建设输送人才,为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是,高等教育质量还需要进一步提高以适应经济社会发展的需要,不少高校的专业设置和结构不尽合理,教师队伍整体素质亟待提高,人才培养模式、教学内容和方法需要进一步转变,学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月,教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》,计划实施"高等学校本科教学质量与教学改革工程"(简称"质量工程"),通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容,进一步深化高等学校教学改革,提高人才培养的能力和水平,更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中,各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势,对其特色专业及特色课程(群)加以规划、整理和总结,更新教学内容、改革课程体系,建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上,经教育部相关教学指导委员会专家的指导和建议,清华大学出版社在多个领域精选各高校的特色课程,分别规划出版系列教材,以配合"质量工程"的实施,满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作,提高教学质量的若干意见》精神,紧密配合教育部已经启动的"高等学校教学质量与教学改革工程精品课程建设工作",在有关专家、教授的倡议和有关部门的大力支持下,我们组织并成立了"清华大学出版社教材编审委员会"(以下简称"编委会"),旨在配合教育部制订精品课程教材的出版规划,讨论并实施精品课程教材的编写与出版工作。"编委会"成员皆来自全国各类高等学校教学与科研第一线的骨干教师,其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求,"编委会"一致认为,精品课程的建设工作从开始就要坚持高标准、严要求,处于一个比较高的起点上。精品课程教材应该能够反映各高校教学改革与课程建设的需要,要有特色风格、有创新性(新体系、新内容、新手段、新思路,教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展,顺应并符合 21 世纪教学发展的规律,代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的"编委会"成员推荐),经"编委会"认真评审,最后由清华大学出版社审定出版。

目前,针对计算机类和电子信息类相关专业成立了两个"编委会",即"清华大学出版社计算机教材编审委员会"和"清华大学出版社电子信息教材编审委员会"。推出的特色精品教材包括:

(1) 21世纪高等学校规划教材·计算机应用——高等学校各类专业,特别是非计算机专业的计算机应用类教材。

(2) 21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

(3) 21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

(4) 21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

(5) 21世纪高等学校规划教材·信息管理与信息系统。

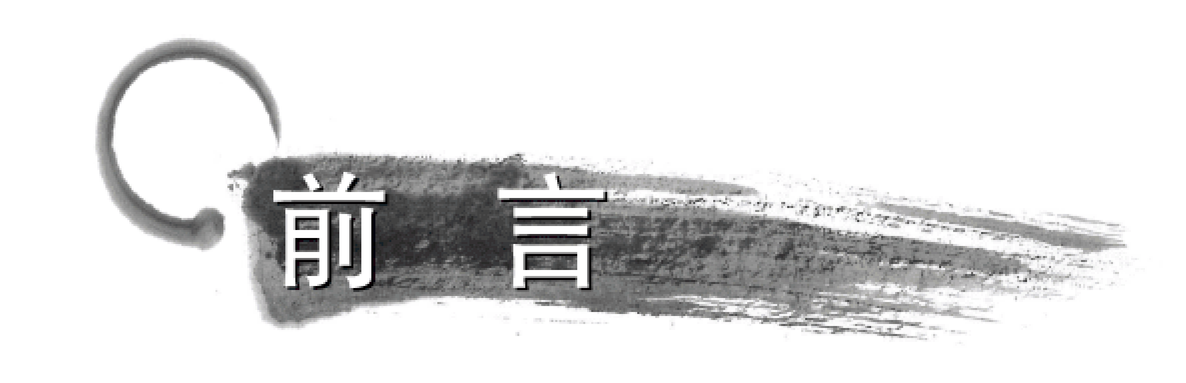
(6) 21世纪高等学校规划教材·财经管理与应用。

(7) 21世纪高等学校规划教材·电子商务。

(8) 21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力,在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌,为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格,这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人:魏江江
E-mail:weijj@tup.tsinghua.edu.cn



写作动机

由于社会需求的推动，软件测试领域近年来非常活跃，涌现了大批优秀的研究成果和经验总结，不仅有各种专门的学术期刊、学术会议、学术组织和每年成千上万的学术论文，还有每年出版的大量新的软件测试教材和著作。

由于一直在软件测试领域从事教学和科研工作，经过多年的积累，一直想写本书。一方面，将自己的积累贡献出来，另一方面，把软件测试领域的知识重新梳理一下，既能让自己的教学和科研水平上一个新的台阶，又能服务社会，为软件测试领域的发展注入一点新的活力。于是，本人便开始实施这项计划，并把这项计划分成三个阶段：第一阶段是调研，包括教材著作调研、学术文献调研和学术期刊会议组织调研、工具软件调研等；第二阶段是对软件测试内各专门领域进行总结，写一批科普性和学术性相统一的学术论文；第三阶段是完成书稿并出版。

内容组织

翻阅了50多本教材和著作，受益匪浅，非常感谢这些作者辛勤的工作，他们确实利用自己的特长和专业积累为软件测试领域的发展作出了多姿多彩的贡献。同时也感到要想在前人的基础上创新，写一本好的专业书籍是非常困难的。

经过长时间的考虑和研究，决定将本书定位于系统地介绍各种软件测试方法。由于近年来发展最为活跃的就是各种软件测试方法，已经出版的教材和著作无法系统涵盖这些新的知识，很多方法还只出现在学术论文之中，处于实验室研究阶段。本书介绍的近90种软件测试方法中，有近一半在我所翻阅的50多本书中根本找不到。

本书在写法上力求简洁、清楚，不仅适合初学者或外行阅读，也要成为从业人员和专家学者的重要工具书和参考书。为此，本书除了第1章软件测试概论之外，基本上按照一个统一的结构介绍每一种软件测试方法。这个框架包括概念、目标、方法、原理、实例、优缺点、研究现状、参考文献等。个别方法因具体情况有所剪裁，特别是有些方法还不是很成熟，只能做一个简单的介绍。

- 目标　针对每一种软件测试方法，首先介绍它的目的，即它可用来测试什么？
- 方法　具体介绍这个方法的内容，即这个方法是什么？
- 原理　介绍方法的原理、理论基础等，即为什么要用这种方法？
- 实例　通过例子将该方法描述得更清楚。
- 优缺点　评论该方法的长处和不足。
- 研究现状　介绍研究现状，为学者和研究人员提供参考(由于篇幅，本书省略了该部分内容)。
- 参考文献　为了写好每种方法，充分参考了已有书籍和学术论文。由于内容比较多，本书只在最后列出了部分典型的参考文献，正文中未进行标注。不当之处，请读者予以指出，将在本书再版时进行修正。

另外为了方便各类学生或读者及时检查学习效果，本书每节后面都配备了习题。

本书在编写过程中得到了南京大学软件新技术国家重点实验室、南京大学计算机科学与技术系、国家自然科学基金(61272079，61021062)、863高科技研究计划(2008AA01Z143)以及江苏省自然科学基金(BK2010372)等单位和基金的资助。

致谢

本人指导的研究生吴化尧、钮鑫涛、梁亚澜、李杰和蒋静等为本书的资料收集、插图的制作做过很多工作，在此表示感谢。

近两年来，围绕软件测试的一些专题，本人指导了一批本科生毕业设计，他们的辛勤工作也为本书编写提供了一些有价值的参考。

编　者

2013年3月


目 录

第1章 软件测试概论

软件测试作为软件质量保障的一种重要的方法，近年来已经得到软件产业界、学术界和软件工程师们普遍重视，目前，软件测试的教学内容已经从以前作为软件工程课程中的一章发展为一门独立的课程，国内外拥有计算机系的大学几乎都开设了这门课程，在国内的几个主要的售书网站上可以找到的各种软件测试的教材和书籍就有40多本。产业界也逐渐认识到不仅软件开发可以成为一门职业，软件测试也是一门重要的职业，人们期待着能够培养出可以尽早发现软件错误的专门人才，这样可以尽量减少软件开发维护的成本和日后因为软件故障而造成的损失。学术界已经把软件测试作为一门重要的相对独立的科学在研究，关于软件测试的各种国际学术会议就有十几个，也有专门的学术期刊和关于软件测试的各种学术组织。本章从软件测试的概念、分类、教育、科研和发展等十几个方面进行简单的回顾，以期对软件测试有一个比较完整的认识。

1.1 软件缺陷与管理

软件测试的核心任务就是要发现软件缺陷与错误。

这里先从错误说起。错误是人们生活的一个组成部分，不仅在思想上、行动上会犯错误，而且这些错误可以表现到人们生产出来的产品上，可以说，错误处处都可以发生。人们可以犯语言错误、观察错误、处方错误、手术错误、驾驶错误、运动错误、恋爱错误等等各种各样的错误，当然程序员在软件开发的过程中也就可能犯类似的错误。错误的后果有时可能微不足道，有时却可能导致重大的灾难，特别是软件开发过程中，人们的错误往往会给软件留下隐患。

程序员在根据设计规格说明编写程序的时候可能会理解错，也可能把代码写错，他们犯的一个或多个错误(Errors)，最后变成程序中的故障(Fault 或 Bug)或缺陷(Defect)，当人们执行程序的时候，就会发现观察到的行为与软件规格说明书中预期的行为之间的差异，这就导致了软件失效(Failure)。

具体来说，软件缺陷可以有以下几种类型：

(1) 软件规格说明书中规定的内容，软件未能实现。

(2) 软件实现了规格说明书中指明不应该实现的内容。

(3) 软件未能实现规格说明书中虽然没有明确说明，但应该实现的内容。

(4) 软件难以理解，不方便使用，可用性差。

软件缺陷管理在很多软件开发组织中是软件开发和测试过程中的一个重要组成部分，它包括缺陷预防、缺陷发现、缺陷记录和报告、缺陷分类和跟踪、缺陷处理和缺陷预测。

(1) 缺陷预防。通过各种过程和工具，如良好的编码技术、单元测试计划和代码审查等预防软件错误。

(2) 缺陷发现。找出软件在静态或动态测试时观察到的错误原因，一般通过调试来完成。

(3) 缺陷记录和报告。发现的缺陷记录在一个数据库中进行管理，数据库记录缺陷的类型、发生的频率、严重程度和发生位置等信息，方便产生缺陷报告。

(4) 缺陷分类和跟踪。数据库中的缺陷一般被分为高严重性和低严重性，并根据缺陷的性质建议处理方式，高严重性错误一般需要尽早提交开发人员处理。

(5) 缺陷处理。每个记录在数据库中的缺陷一开始都被标为开放状态，表示需要进行处理。必须指定一些开发人员来进行缺陷的定位和修改工作，直至缺陷消除，这时将数据库中的缺陷状态置为关闭，表明缺陷已经处理好。

(6) 缺陷预测。通过一些高级的统计模型，根据已有的缺陷信息和统计数据，对软件中可能存在的错误数量和性质进行预测，是软件缺陷管理的一个重要功能。

目前关于缺陷管理的工具有很多，例如，开源的工具 Bugzilla、商业的工具 FogBugz。

习题 1.1

1. 软件错误有哪几种类型?
2. 缺陷管理具体包括哪些功能?

1.2 软件质量

软件测试的核心目的就是要度量和提高软件质量，那么什么是软件质量?

软件质量是一个包含多个属性的，可度量的多维度的量，其中有些属性比另外一些属性重要。根据软件质量的度量方式可以分为静态质量属性和动态质量属性，静态质量属性是指实际代码和相关文档的质量，包括代码的结构化、可维护性、可测试性，文档的完整性和可读性；动态质量属性是指软件使用过程中质量属性，包括软件的可靠性、正确性、完备性、可用性、可维护性、可信性和性能等。

由于软件是一种不可见的复杂的逻辑体，而且软件开发过程中环节较多，测试工作上的人力、物力是有限的，不可能做穷举测试，因此，客观上给软件质量留下很多隐患。再加上我们在技术上也存在很多局限性，例如，软件测试工具本身也需要测试、软件测试工具对软件的理解程度达不到人的理解程度、软件技术的发展使得软件的测试方法需要不断地扩展新的测试技术，以及软件质量没有统一指标来评价等因素，在度量和提高软件质量方面，我们还有很长的路要走。

习题 1.2

1. 查阅相关文献，给出一个相对比较完整的软件质量度量指标体系。

2. 软件质量体系中很多指标是相互矛盾的，请举出几个例子。

1.3　软件测试的定义、性质及相关概念

1. 软件测试的定义

软件测试的定义反映了人们对该领域的认识，到目前为止，关于软件测试的定义经历了一个从简单到复杂的、从单一化到多元化的过程。以下以时间为序，列举一些有代表性的定义，这些定义之间有的可以相互补充，有的可以相互包含，各具特色，可以从不同角度对软件测试做出比较全面的和准确的表述。

定义 1　测试是为发现错误而运行一个程序或系统的过程。

定义 2　测试是以评价一个程序或者系统的属性为目标的任何一种活动，测试是对软件质量的度量。

定义 3　使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果与实际结果之间的差别。

定义 4　软件测试是软件生存周期内的一个重要阶段，是软件质量保证的重要步骤，是软件投入运行前，对软件需求分析、设计规格说明和编码进行最终复审的活动。

定义 5　测试是为了度量和提高被测试软件的质量，对测试软件进行工程设计、使用和维护的并发生命周期活动。

定义 6　软件测试是一种作为主体的人通过各种手段对客体软件的某种固有属性进行的一种以认知和改造为目的的活动。

定义 7　软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计的一批测试用例，并利用这些测试用例运行程序以及发现错误的过程。

定义 8　软件测试以寻找软件错误、增强软件正确性和满意度等为目标，对软件进行测试设计、生成、执行、结果记录和分析维护等活动，在此过程中坚持预防性为主、充分进行风险分析、尽量使用工具等原则，以提高测试活动的科学性、有效性和经济性。图 1-1 给出的软件测试的三个层面，即目标、活动和原则，强调三维一体。

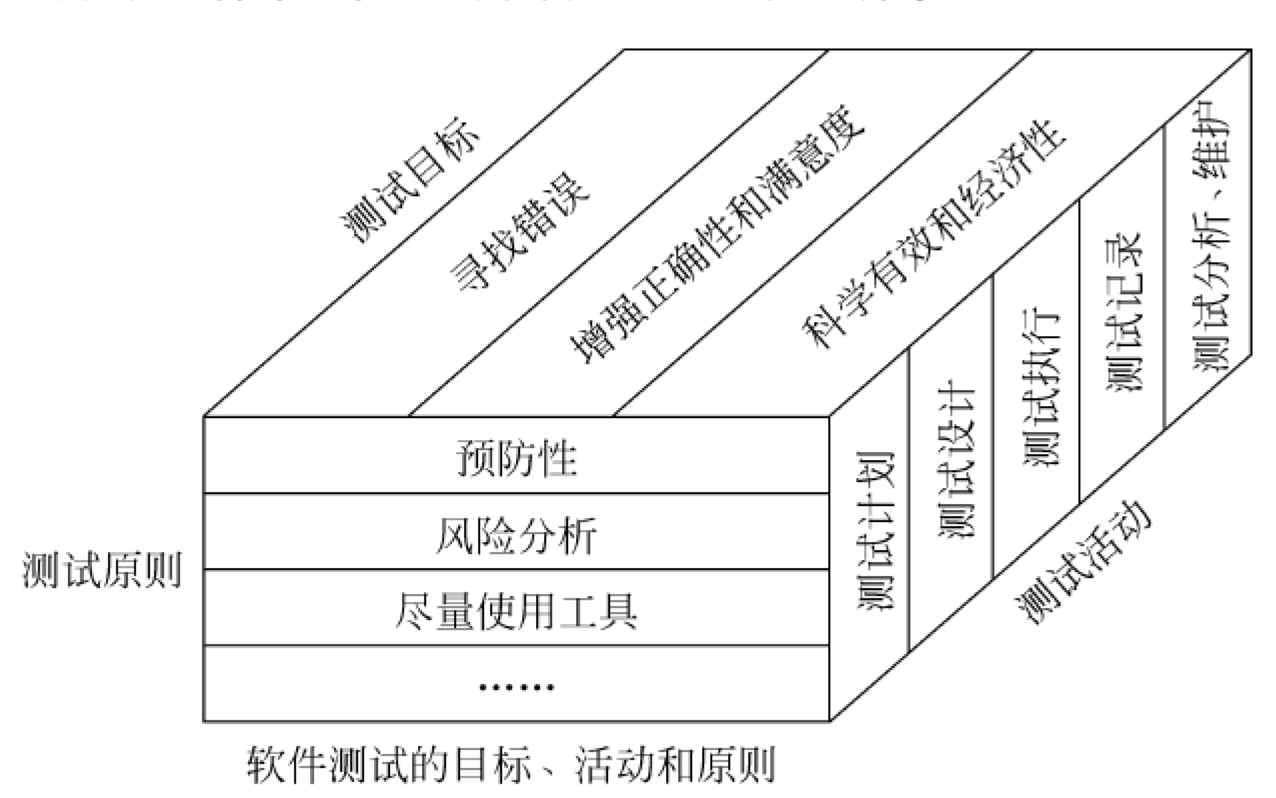


图 1-1　软件测试的三个层面

以上这些定义体现了两个方向：一个是越来越抽象，即概念的内涵越来越小，外延越来越大，本书列出的几个定义中最抽象的是定义7，根据这个定义实际上人类的很多活动都属于这个定义范围，所以外延很大；另外一个方向是越来越具体，最具体的是我们给出的定义8，对软件测试的很多方面都进行了具体的规定，非常的具体。这些定义体现人们的认识过程和深度，也适合不同的读者口味，把这些定义列在这里，可以帮助读者充分理解软件测试的概念，为以后的学习打下基础。

2. 软件测试的性质

(1) 存在性或永不过时性。任何软件不可能不存在缺陷，而且无论软件开发技术如何发展，不可能有一天，因为高度发达的软件开发技术，在软件开发过程中及软件开发出来之后，软件不需要进行测试，就可以直接使用。所以在任何时候，只要人们还需要软件这种产品，只要还有软件开发，就会需要软件测试，软件测试将因为软件的存在而存在。

(2) 不完全性或抽样性。一般情况下，软件的输入空间非常庞大，程序中可以执行的路径也可能无法穷尽，而且软件的规格说明有着不同标准，可能存在不完整、不准确等问题，这就决定了软件测试是一种不完全测试、部分测试或者抽样性测试。

(3) 证错不证对性。不完全的软件测试无法证明软件的正确性，只能证明软件的不正确性。尽管软件测试是保证软件质量的一个重要手段，但它的能力是有限的；经过软件测试的软件不一定就完全可靠。

(4) 检错保质性。尽管软件测试的定义可以有多样，且形式和内容有别，但有一点可以确定：软件测试与软件质量不可分割，软件测试始终与软件缺陷的发现过程密切相关。

(5) 综合性或多方求证性。软件测试是一项技术性工作，同时涉及经济学和心理学的一些重要因素，甚至涉及管理学的很多内容，所以，软件测试是一门实践性很强的综合学科。

(6) 测试效果评判准则。一个好的测试用例在于它能发现至今未发现的错误；一个成功的测试是发现了至今未发现的错误的测试。

3. 软件测试的相关概念

(1) 软件测试的对象。软件测试并不等于程序测试，它贯穿于软件定义与开发的整个过程，包括需求分析、概要设计、详细设计以及编码等各个阶段所得到的文档，所以需求规格说明、概要设计规格说明、详细规格说明以及源程序都是软件测试的对象。

(2) 测试用例。为特定的测试目的而设计，通常包括一组测试输入、执行条件和预期的结果，其中预期结果除了预期输出，还可以指定一些其他观察要求，如各种测试覆盖率等；测试用例是执行测试的最小实体。

习题 1.3

1. 综合本节中给出的关于软件测试的定义，给出一个比较完整的软件测试概念描述。
2. 软件测试具有哪些特性？
3. 软件测试的对象包括哪些方面？
4. 什么是测试用例？
5. 以一个身边常用的具体软件为例，如手机软件或播放器软件，诠释本节中提到的几

个概念，包括软件测试、测试对象、测试用例以及测试的若干特性。

1.4 软件测试的目标

测试目标是软件测试追求的目的，是软件测试活动存在的理由，笼统地讲就是为了度量和提高软件质量。具体来讲，软件测试目标包括：发现软件缺陷，特别是在软件发布之前发现软件缺陷，从而提高软件质量，树立人们对软件质量的信心，软件开发商才能在激烈的市场竞争中有竞争力；验证软件功能、评测软件的性能以及提高软件可靠性，验证软件已正确实现了用户的要求等。

软件测试的目标还可以从验证和确认两个方面来进行理解。验证（Verification）是指在软件生命周期的各个阶段，用下一阶段的产品来检查上一阶段的规格定义。例如，通过设计来验证需求定义的规格是否正确，通过编码来验证设计的合理性，通过测试来验证编码的正确性。确认（Validation）是指在软件生命周期的各个阶段，检查每个阶段结束时的工作成果是否满足生命周期的初期在需求文档中定义的各项规格和要求。例如，在软件设计完成后，需要通过评审来判断是否满足需求定义；编码完成之后，也需要通过代码审查等方式来检查是否满足各项需求的规格定义；在测试阶段，通过评审测试用例、测试计划、测试文档、测试报告、发现的缺陷等材料来判断测试是否覆盖了各项需求。

总之，测试的目标就是以最少的时间和人力，系统地找出软件系统中潜在的各种错误和缺陷。如果成功地实施了测试，就能发现软件中的错误。

在测试过程中，软件测试人员的能力是以其能够发现软件中潜在错误的能力来衡量的，这正如病人到医院看病的例子。病人到医院看病，医生应该先做检查，找出哪里有病，然后推荐治疗措施。如果这个医生经常是检查了半天，没有发现任何问题，说明这个医生医术不高。同样，待测试软件是一个病人，测试人员是医生，怎样才能找出问题，这完全要依赖测试人员的修行。医生能够越老越值钱，是因为他们一直战斗在救死扶伤的第一线，积累了丰富的医疗经验。同样，如果测试人员一直坚持工作在测试第一线，理论联系实际，不断地积累实践经验，不断地学习和总结，他们应该越来越有价值。

习题 1.4

1. 软件测试的目标是什么？
2. 验证（Verification）和确认（Validation）的区别和联系在哪里？
3. 作为软件测试的从业人员如何在日常生活和工作中提高个人能力和水平？

1.5 软件测试的意义和重要性

从开发方角度，软件测试可以尽早发现和改正软件中潜在的错误，不仅避免给用户造成可能的损失，更重要的是避免给开发方的信誉造成不良影响，即避免最终给开发方自己造成直接和间接的损失。

对于用户来说，适当的测试是用户对是否接受软件系统的依据，用户通过亲自的测试或

者委托第三方测试的结果来对软件系统的质量作出判断,并最后作出选择。

近年来,软件缺陷给人们造成重大损失的例子数不胜数,表 1-1 简要地列出了一些影响比较大的事件,关于事件的完整故事,感兴趣的读者可以查阅有关软件测试教材和资料。

表 1-1 典型的因为软件缺陷造成重要损失的案例

	事件名	时间	原因	后果
1	狮子王游戏事件	1995	缺乏充分的配置测试	客户无法使用,退货损失
2	英特尔奔腾芯片缺陷	1994	浮点数除法算法缺陷	支付 4 亿美元更换芯片
3	放射性设备医疗事故	1985	软件设计存在缺陷	多人因超剂量辐射死亡
4	Norton“误杀门”	2007	将系统文件定义为病毒	超过百万台电脑瘫痪
5	NASA 火星登录器	1999	集成测试不充分	登录器丢失
6	爱国者导弹事故	1991	计时器误差	误杀 28 名美国士兵
7	千年虫问题	2000	节省内存,时间表示的缩写	全球损失几千亿美元
8	Windows 2000 安全漏洞	2000	操作系统远程服务软件存在安全漏洞	拒绝服务、权限滥用、信息泄露
9	阿丽亚那火箭爆炸	1996	惯性导航系统软件故障	9 年的航天技术受重挫
10	“冲击波”计算机病毒	2003	微软 Messenger Service 缺陷被攻破	成千上万台基于 Windows 的计算机崩溃
11	北京奥运会售票事件	2007	大量访问造成网络拥堵	无法售票
12	F-16 战机失事	2007	导航软件失灵	飞机失事
13	美国及加拿大停电	2003	电力管理系统故障	大规模停电
14	金星探测器飞行失败	1963	程序语句中缺少一个逗号	损失 1000 多万美元
15	新西兰客机撞山	1979	飞行软件故障	257 名乘客全部遇难
16	伦敦救护中心瘫痪	1992	计算机系统故障	整个机构无法工作
17	北京机场瘫痪(经常)	2005	软件系统故障	飞机无法起飞,旅客滞留
18	银行系统故障(经常)	2006	软件系统故障	无法工作
19	校园网瘫痪(经常)	2000	软件系统故障	无法工作
20	金山词霸出现的错误	2003	系统安装问题	无法取词,无法解释

习题 1.5

1. 软件测试具有什么重要的意义?
2. 请举几个读者身边或了解的案例,说明软件故障给人们带来的危害和损失。

1.6 软件测试的 20 条原则

(1) 软件测试应追溯到用户需求。软件开发的最终目的是要满足最终用户的需求,图 1-2分析了软件缺陷从原始需求开始,经历需求分析、设计和编码阶段逐步形成的过程,从该图可知,要完成对待测试软件有效的测试,测试人员要紧紧围绕用户需求,站在用户角度看问题,系统中最严重的错误是那些导致无法满足用户需求的缺陷。用户是软件的使用者和投资者,不满足用户需求的软件是无法顺利交付和收回成本的。

(2) 软件测试应该尽早开始,不断进行。由于软件的复杂性和抽象性,以及涉及项目人员之间沟通不畅等原因,导致在软件生命周期的各个阶段都可能产生缺陷(如图 1-2 所示)。

软件错误发现得越早,修改错误的成本就越低,所以测试要从需求分析和设计就开始,在软件开发的每个环节都要不断地进行测试,才有可能使错误不被遗漏。极限编程(XP)和测试驱动的开发(TDD)将“尽早开始,不断进行”用到极致,它们强调测试先行,在编码开始之前就开始撰写测试,用测试指导代码的撰写。目前 XP 和 TDD 正受到越来越多人的关注。

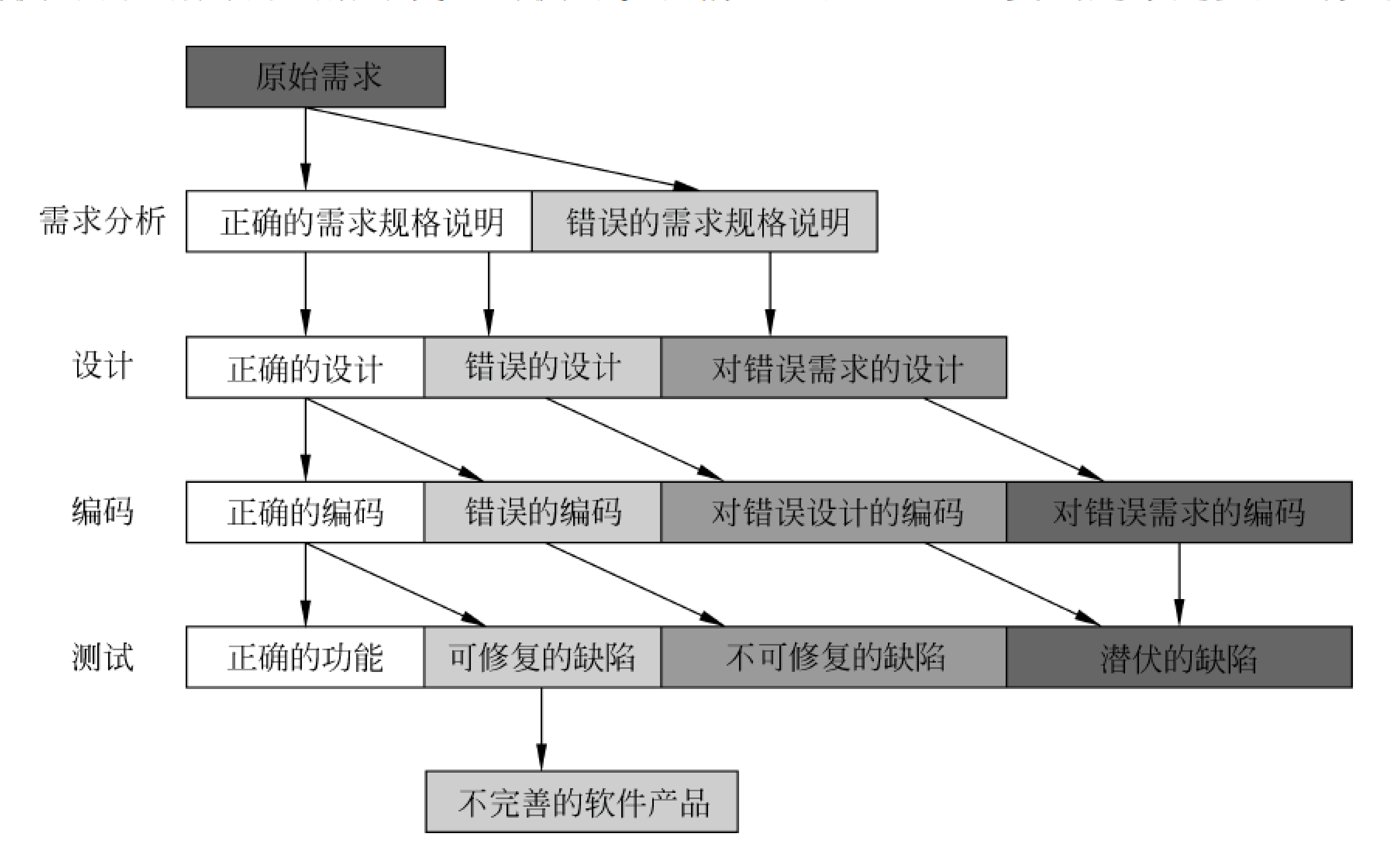


图 1-2 软件缺陷形成过程示意图

(3) 穷尽测试是不可能的,测试要有停止准则,适可而止。完美的情况下,通过在有限的时间和资源条件下,穷尽运行软件,发现所有问题。但实际上是不可能的,这可以从软件的基本要素——输入、输出、数据处理和计算等方面来分析。因为程序输入一般都非常大,无法计数,输出也很复杂,有时甚至超过输入的复杂度。数据处理和计算的复杂程度也越来越高,导致路径组合近似天文数字。因此,只能通过多种测试方式,在有限资源下,尽可能多地发现缺陷。测试停止准则有很多,大致可以分为 5 类:①测试超过了预定时间,则停止测试;②执行了所有的测试用例,但并没有发现故障,则停止测试;③使用特定的测试用例设计方案作为判断测试停止的基础,例如代码覆盖、路径覆盖等各种覆盖率等;④正面指出停止测试的具体要求,即停止测试的标准可定义为查出某一预订数目的故障;⑤根据单位时间内查出故障的数量决定是否停止测试。Good Enough 原则指出:测试的投入与产出要适当权衡,测试进行的不充分是对质量不负责任的表现,但是投入过多的测试,则是资源浪费的表现。

(4) 一个测试用例除了要对软件的输入数据进行详细描述,还要给出预期输出结果。否则,在测试过程中,极有可能因为无法与预期输出进行比对,而使应该发现的错误从眼皮底下溜走,降低测试效果。

(5) 程序员应该避免测试自己的程序,软件测试应该具有一定的独立性。待测试程序就好像程序员自己生的孩子,怎么看怎么顺眼,客观上,程序员很难跳出自己的思维习惯,对自己编写的程序做科学的测试。软件测试应该是“破坏性的”,以“找茬”为目的的,独立的软件测试才有可能发现意外的错误。而且软件测试中存在一种“同化效应”,即一名测试人员在同一个项目待的时间越久,越可能忽略一些明显的问题,例如对于界面操作,由于测试人

员重复使用一个软件而产生熟悉感，因此可能会忽视一些易用性问题和用户体验问题。同时测试人员与开发人员在一起工作时间较长之后，容易被同化。同化效应会使 Bug 具有免疫效果，因此，测试过程中还需要通过轮换或补充新测试人员来避免同化效应。

(6) 应该彻底检查每个测试的结果。每个测试用例一般都是经过精心设计产生的，承担一定的检测任务。由于测试执行结果可以形式多样，如界面的显示、数据文件以及性能测试中关注的软件执行时间等，不同形式的结果对于判断测试是否通过具有不同程度的影响。而且软件的预期输出描述可能不是很准确，这就更需要认真检查每个测试结果。

(7) 测试用例的编写不仅应该包括有效(合法)和预料到的输入情况，还要考虑无效(非法)和未预料的输入情况。程序员往往基于预料之中的合法输入编写程序进行处理，可能忽视对一些非法输入或意料之外的输入的处理，从而使程序留下隐患。

(8) 不仅要检查程序是否"做了它应该做的"，而且要检查程序是否"做了它不应该做的"。程序只能做被规定的那些工作，任何多余的功能都应视为安全隐患。全面的测试包括：验证程序是否做了应该做的事情，以确保软件的可靠性；验证程序是否没有做不应该做的事情，以确保软件的安全性。

(9) 测试用例要进行管理和保存，切忌用后即弃，要重视测试过程的记录并保存相关文档。除了一次性软件，几乎所有的软件测试都不可能一次性完成，在软件生命周期中一般都要经过测试、修改、再测试、再修改等多次反复。原先的测试用例可以为再测试的选择性复用、生成新测试用例和测试分析等提供非常好的依据和基础，避免重复劳动，特别是在回归测试中，要使用原有的测试用例检查修改是否引入新的错误。

(10) 程序中的错误存在"群集现象"。程序中如果某处发现错误，在该处和周围还可能存在错误，有人甚至认为："程序中某部分存在错误的可能性，与该处已发现错误的数量成正比"。人们也用"冰山原理"和"二八原理"来定性程序中错误的分布。所谓"冰山原理"，就是两座冰山，露出水面部分大的那一座，其水下部分必然也较大。"冰山原理"与潜在错误与已发现错误成正比关系的表述相一致。所谓"二八原理"，来自社会学观点"20%的富人，掌握了 80%的社会财富，其他 80%的人只占有 20%的社会财富"，在软件工程中该原理也多次被证明是成立的，例如"20%的代码，包含了软件中 80%的缺陷"，"20%的缺陷，消耗了 80%的软件修改维护费"。"二八原理"与软件错误的"群集现象"的表述相似。

(11) 选择一组合适的测试方法。软件系统的不同部分具有不同的特点和不同的重要程度，需要不同的测试方法才能充分的有针对性的测试，因此，在测试过程中选择合适的测试方法也非常重要。

(12) 测试应该认真做好测试计划，使用测试清单。测试计划和清单列出所有需要测试的对象、场景和条件列表，提供了一种确定测试覆盖率的方法，并可以作为软件测试质量和软件质量的依据。测试过程中，切忌测试的随意性。

(13) 恰当使用测试工具。工具可以延伸人的能力，很多测试工作，如测试数据生成，测试结果的比对，如果手工完成，既很累人又容易出错。测试工具的正确使用不仅能够有效提高测试效率，而且也可以提高正确率，使测试人员有更多精力从事创造性测试工作。

(14) 不断进行培训。测试作为一门学科，和其他学科一样，有着一套知识体系和特定的技能要求。另外，根据一些测试人员执行的测试类型，甚至还需要特殊的或先进的知识和技能。测试人员只有不断地进行培训或自我培训，才能适应测试工作中新的挑战。

(15) 进行软件风险分析,做有重点的测试。由于不可能进行穷尽测试,某种意义上,测试了什么比测试了多少更重要。软件风险分析可以对测试对象进行优先级排序,这样合理分配测试资源,集中精力对软件系统中关键的部分进行充分测试。

(16) 度量测试有效性,对出错结果进行统计和分析。使用科学的度量可以使测试工作更加科学,例如,对各种测试方法有效性的度量,可以帮助人们选择有效的测试方法;对于测试人员测试效果的度量,可以帮助人们判断哪些测试人员更有工作潜力。同时应对测试过程中找到的所有缺陷进行定性分析和预测,以决定是否进入下一个测试阶段。

(17) 分析缺陷趋势和模式。缺陷之间往往存在一些相互关联和依赖关系,通过分析测试人员发现的缺陷的趋势、模式、类型、来源、严重程度和潜伏期等,可以进一步发觉一些新的问题,从中找到需要改进地方。

(18) 增量和分级测试。测试范围应从小规模开始,逐步转向大规模的增量和分级的测试。所谓小规模是指测试的粒度,如某种程度的单元测试,进一步的测试从单个单元测试逐步过渡到多个单元的组合测试,即集成测试,最终过渡到系统测试。

(19) 软件测试是一项极富创造性、极具智力挑战性的工作。不仅测试一个大型软件所需要的创造性很可能超过了开发该软件所需要的创造性,而且要充分测试一个软件以确保所有错误都不存在是不可能的,因此,这就给软件测试的从业人员留下了开阔的创造性空间,并使软件测试成为一门职业成为可能。

(20) 软件测试是一种服务。软件测试人员通过对软件产品进行研究和探索,获取有关软件的信息,供项目决策者作出正确的决定。特别是敏捷开发中,软件测试指引着团队的工作。把软件测试理解成一种服务可以化解测试人员和开发人员之间的矛盾,从而有利于客观公正地进行测试工作。

习题 1.6

1. 在本节所列出的 20 条原则中,请给出读者认为的软件测试应该遵循的最重要的几条原则,简要说明理由。

2. 除了本节中提到的 20 条原则,读者认为软件测试还应遵循哪些原则? 例如,软件测试应该是动态自适应的,即软件测试应该能够随着测试过程进行动态和适应性调整。

1.7　软件测试的分类

本节主要介绍软件测试的分类以及分类后分别有哪些测试方法。软件测试课程最重要的知识体就集中在各种测试方法上,这门课程的后续部分就是围绕软件测试的分类,介绍各类软件测试方法的工具和管理等。

(1) 软件测试根据是否运行程序可分为静态测试和动态测试,静态测试包括桌面检查、代码审查和代码走查等方法。动态测试根据测试用例设计是否依据内部结构可以分为黑盒测试和白盒测试,白盒测试包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖、线性代码序列及跳转测试等;黑盒测试包括等价类划分、边界值分析、因果图分析、错误猜测、状态转换测试等。

(2) 根据软件开发的不同阶段可以将软件测试划分为单元测试、集成测试、系统测试、验收测试、回归测试、验证测试、确认测试、α 测试、β 测试和 γ 测试等。

(3) 根据被测试软件的开发方法和应用环境的不同可以分为面向对象软件测试、面向方面软件测试、面向服务软件测试、构件软件测试、嵌入式软件测试、Web 应用软件测试等，后面还要出现普适计算环境下的软件测试、云计算环境下的软件测试等。

(4) 根据软件不同特性和方面测试可以分为：负载测试、压力测试、性能测试、安全性测试、安装测试、可用性测试、稳定性测试、授权测试、用户接受性测试、一致性测试、配置测试、文档测试、兼容性测试和试玩测试(Playtest)等。

(5) 根据不同特殊的测试技术可以有：组合测试、蜕变测试、变异测试、演化测试、模糊(FUZZ)测试、基于性质的测试、基于故障的测试、基于模型的测试、基于操作剖面的测试、基于用例和/或用户陈述开发测试用例、基于规格说明的测试、统计测试、逻辑测试、随机测试、自适应随机测试、GUI(Graphical User Interface，图形用户界面)测试、冒烟测试和探索测试等。

习题 1.7

1. 软件测试方法可以有几种分类方法？

2. 根据本节分类方法，面向软件新技术的测试方法和面向新型运行平台的软件测试方法属于哪一类方法？

3. 软件测试方法与其他产品测试方法有哪些区别和联系？

1.8 软件测试的过程及策略模型

人们在软件开发实践中总结出许多开发模型，如瀑布模型、快速原型法、螺旋模型等，软件测试是其中的一个重要组成部分。随着人们对测试的重视程度与日俱增，人们认识到以往的开发模型都未强调测试的作用，不能充分指导测试实践。软件测试与软件开发一样，包含一系列有计划的系统性活动，其流程和方法也应有相应模型的指导。人们在软件测试活动的实践中总结出了一些很好的测试过程模型，这些模型对测试活动进行抽象，并与开发活动有机结合，可有效提高测试过程的质量，提高测试结果的有效性和准确性。下面将分别介绍软件测试基本过程、在各种开发模型中的位置和人们提出的一些新的测试过程策略模型。

1. 软件测试基本信息流模型

软件测试基本信息流模型如图 1-3 所示。

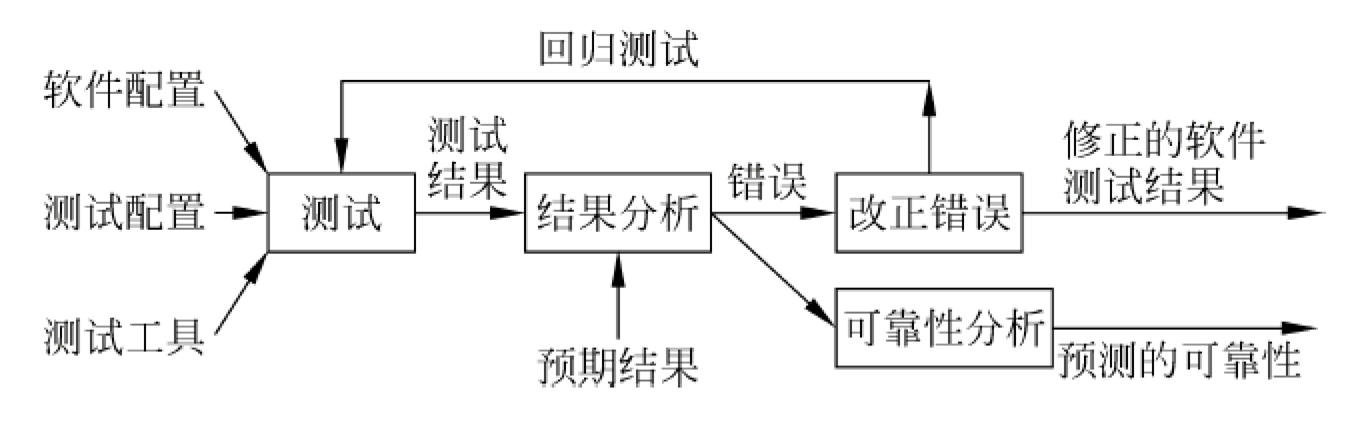


图 1-3 软件测试基本信息流模型

(1) 软件配置。是指测试对象,它包括软件需求规格说明、软件设计说明和被测试的源程序清单。

(2) 测试配置。包括测试计划、测试用例、测试驱动程序,实际上,在整个软件工程中,测试配置只是软件配置的一个子集。

(3) 测试工具。为提高软件测试效率,可使用测试工具支持测试。其作用就是为测试的实施提供某种服务,以减轻人们完成测试任务中的手工劳动。例如,测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序以及驱动测试的数据库等。

2. 软件测试在开发模型中的位置

(1) 在瀑布模型中的软件测试。传统的瀑布模型是将软件生命周期的各项活动规定为按照固定顺序相连的若干个阶段性工作,形如瀑布流水,最终得到软件产品。优点包括:易于理解;调研开发的阶段性;强调早期计划及需求调查;确定何时能够交付产品及何时进行评审与测试。缺点包括:需求调查分析只进行一次,不能适应需求变化;顺序的开发流程,使得开发中的经验教训不能反馈到该项目的开发中去;不能反映出软件开发过程的反复与迭代性;没有包含任何类型的风险评估;开发中出现的问题直到开发后期才能够显露,因此失去及早纠正的机会。改进的瀑布开发模型如图 1-4 所示,人们在每一步中增加了向前回溯和修正的步骤。

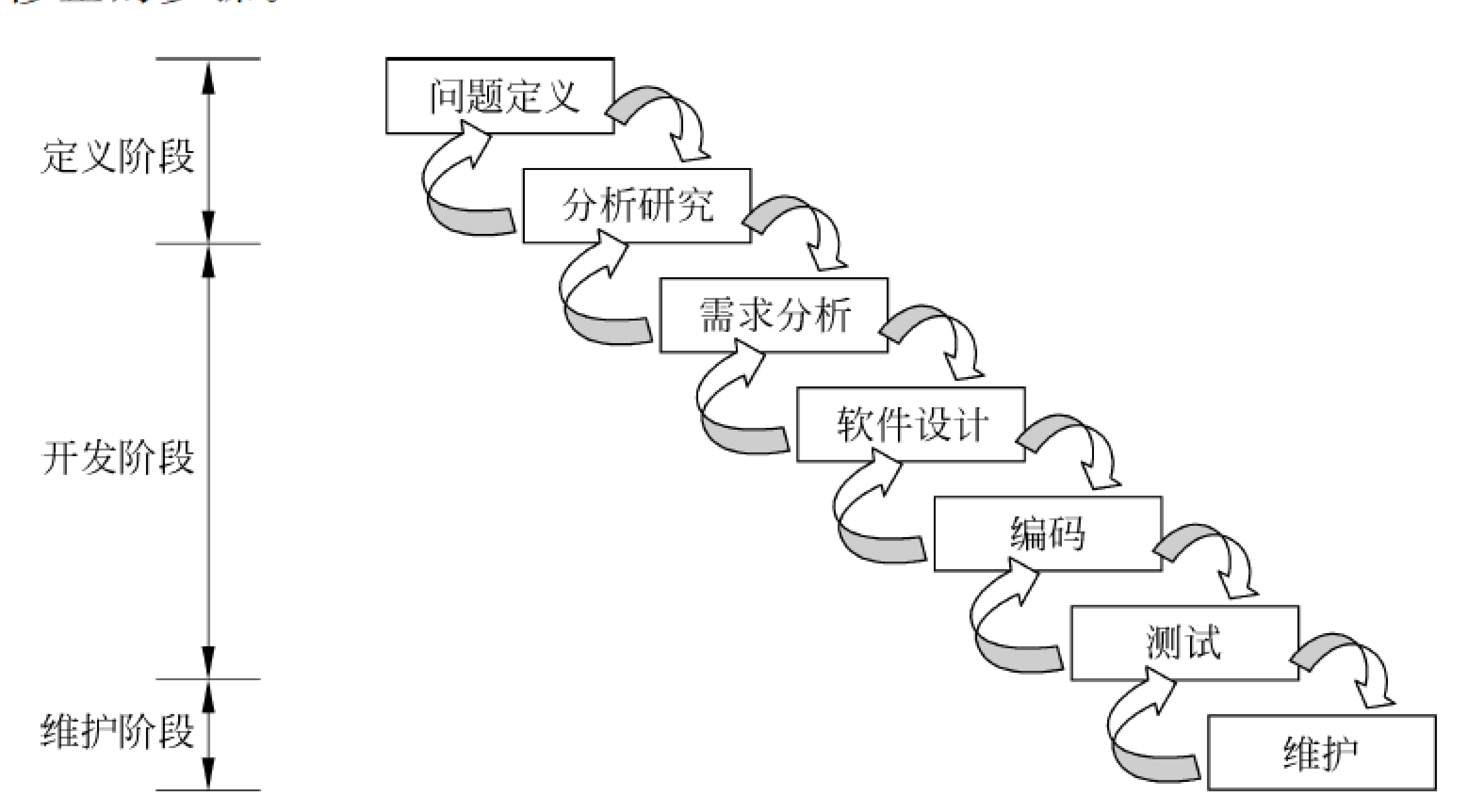


图 1-4　瀑布开发模型

(2) 在快速原型模型中的软件测试。根据客户需求在较短的时间内解决用户最迫切解决的问题,完成可演示的产品。这个产品只实现最重要功能,在得到用户的更加明确的需求之后,原型将丢弃。在该软件开发模型中,软件测试没有明确位置,只能理解为隐含在原型开发、原型评价和系统实现当中。快速原型开发模式如图 1-5 所示。

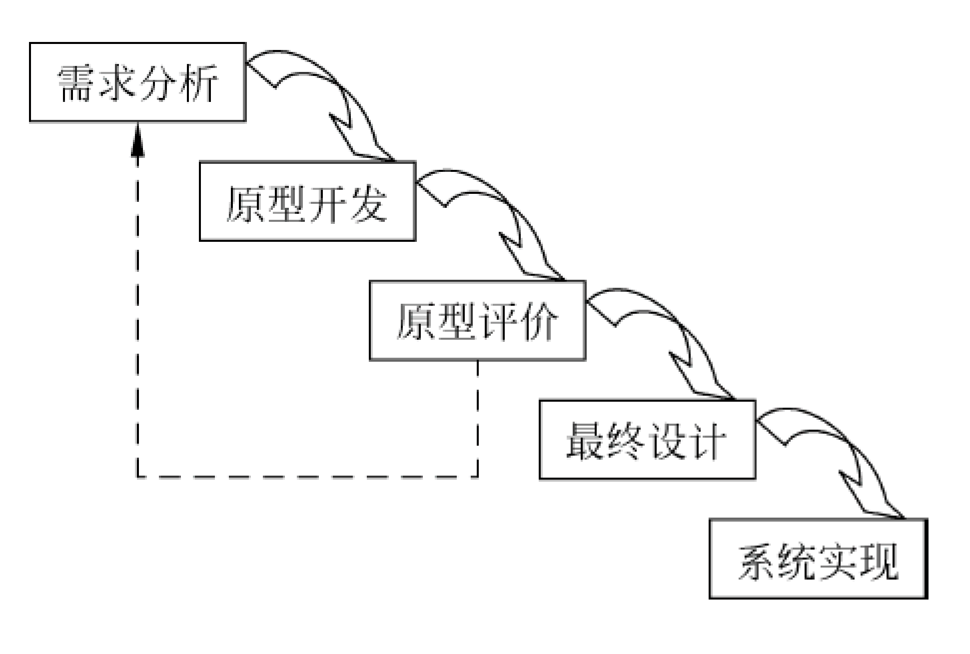


图 1-5　快速原型开发模型

(3) 在螺旋开发模型中的软件测试。螺旋模型将瀑布模型与快速原型模型相结合,并加入风险评估所建立的软件开发模式。主要思想

是在开始时不必详细定义所有细节，而是从小开始，定义重要功能，尽量实现，接受客户反馈，进入下一阶段，并重复上述过程，直到获得最终产品。每一螺旋(开发阶段)包括5个步骤：①确定目标，选择方案和限制条件；②对方案风险进行评估，并能解决风险；③进行本阶段的开发和测试；④计划下一阶段；⑤确定进入下阶段的方法。优点包括：严格的全过程风险管理；强调各开发阶段的质量；提供机会评估项目是否有价值继续下去。螺旋开发模型如图1-6所示。

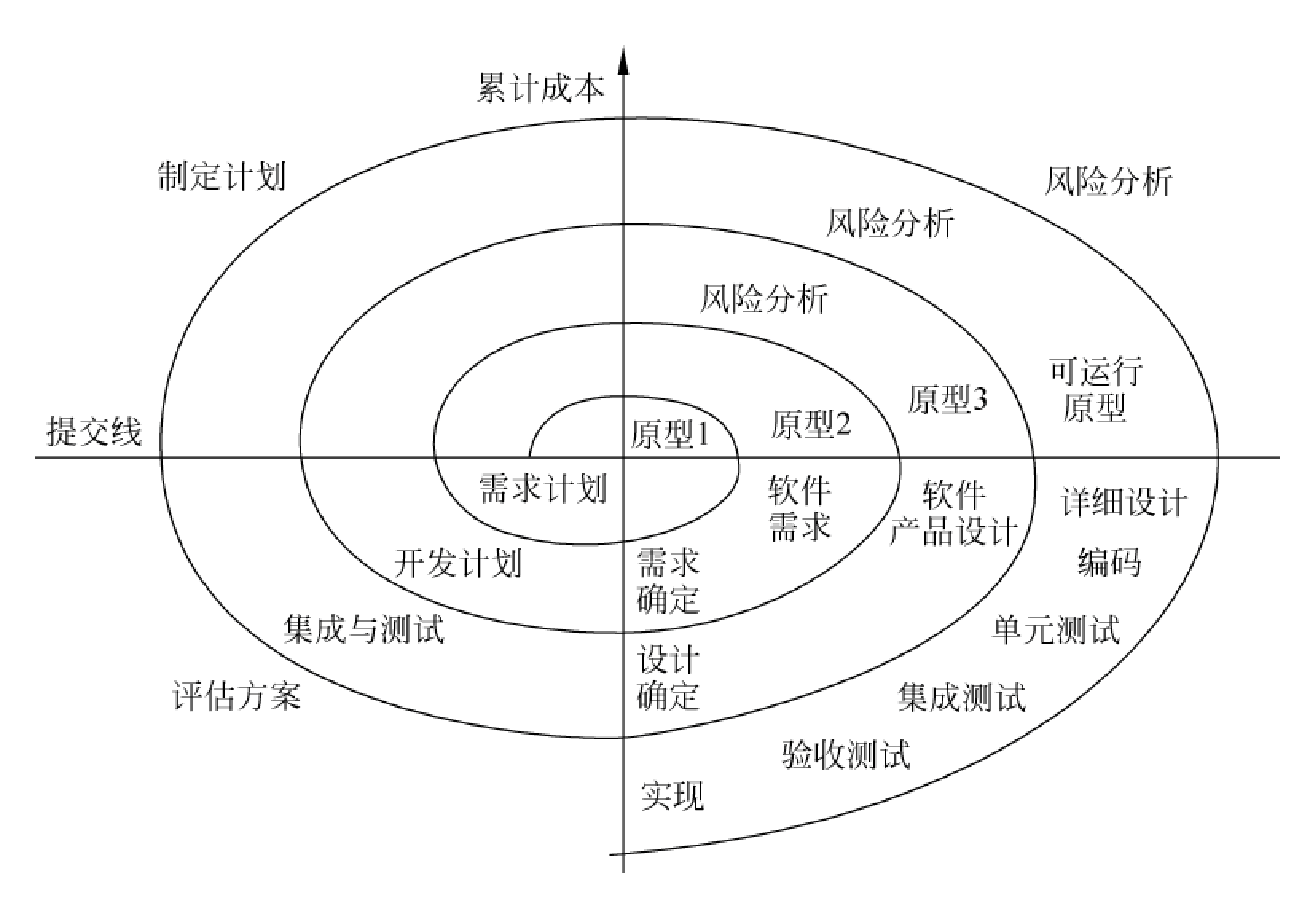


图1-6 螺旋开发模型

3．软件测试过程及策略模型

(1) V模型。V模型如图1-7所示。

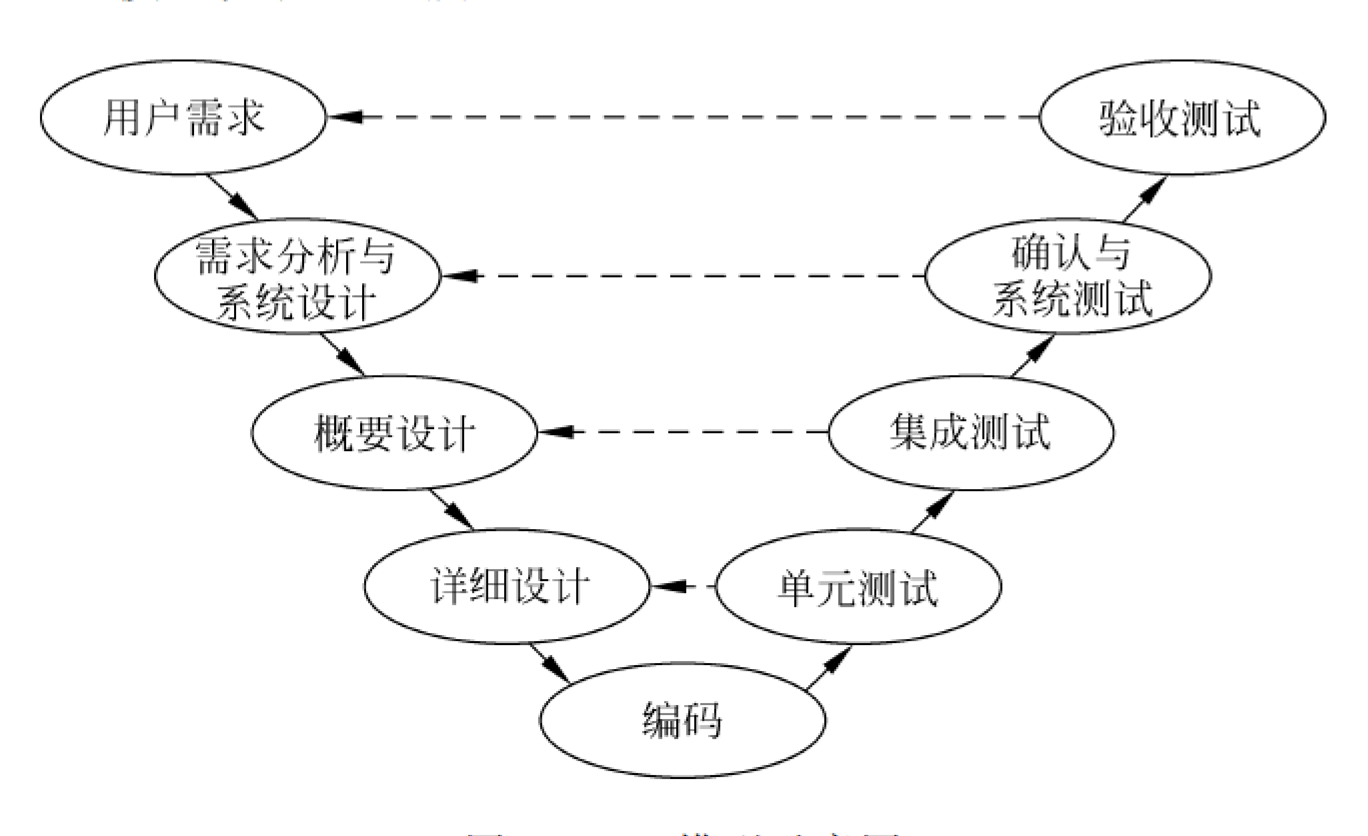


图1-7 V模型示意图

1980年后期，Paul Rook在软件开发的瀑布模型的基础上提出软件测试V模型，旨在改进软件开发与测试的效率和质量。图中箭头代表时间方向，左半部分描述基本的开发过程，开发从上到下，严格地分为不同的开发阶段；右半部分描述了与开发相对应的测试

过程，自下而上，也严格分为不同阶段和级别，测试活动的展开次序正好与开发的次序相反，且动态的测试行为与开发行为相对应，每个测试阶段的基础是对应开发阶段的提交文档。

V模型测试策略的底层测试可以保证源代码的正确性，高层测试保证整个系统满足用户需求。它的不足是：该模型将测试置为编码之后的一个阶段，未能体现"尽早测试，不断测试"的原则。在实际情况下，各级测试与开发各个阶段的文档很难有完美的一对一关系，例如，需求分析的文档往往不足以为系统测试提供足够的信息，经常需要借助概要设计和详细设计文档中的部分内容。在V模型中缺少需求评审、代码审查等静态测试内容。而且测试时间往往因前期开发进度的拖延而被挤占，甚至取消部分测试，导致测试不充分，大大降低测试质量。

(2) W模型。为解决V模型的不足，贯彻"尽早测试、不断测试"的原则并体现静态测试，Paul Herzlich提出了W模型，即开发过程是一个V，伴随的测试过程是另一个V，是并行关系，如图1-8所示，每个开发行为对应一个测试行为，例如，开发行为是各种需求定义和编码，则对应的测试行为是对这些文档的静态测试，开发行为对应的是软件实现，则对应的测试行为是对这些实现的动态测试。

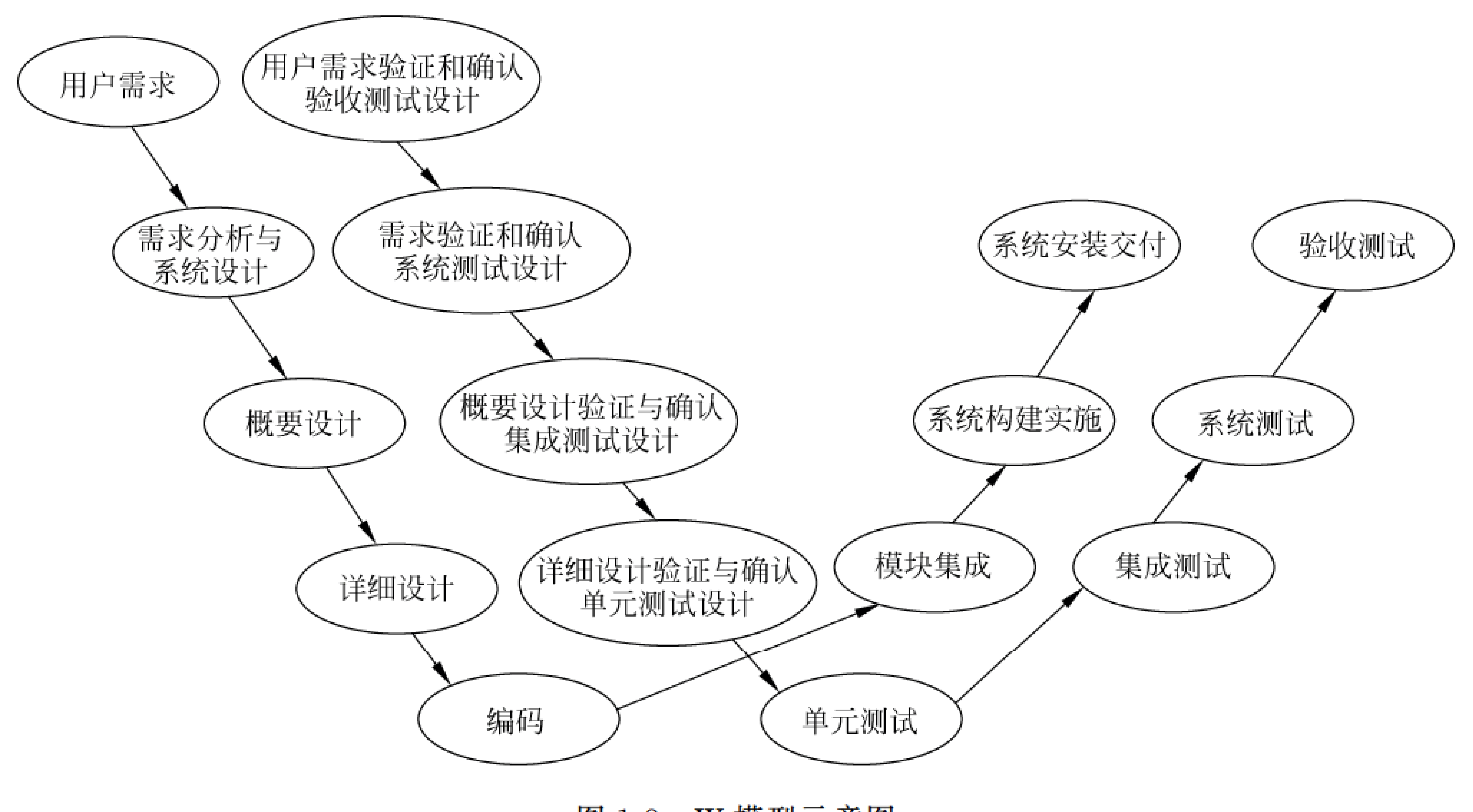


图1-8 W模型示意图

W模型的不足是将软件开发和测试都看成是串行的活动，保持着线性关系，无法支持迭代的开发模型，不支持变更的调整，未体现测试流程的完整性。

(3) H模型。为克服V模型和W模型均将开发和测试看成串行活动，无法体现完整测试流程的缺陷，人们提出H模型，H模型将测试活动完全独立出来，形成完全独立的流程，该流程分为两个阶段：测试准备阶段(包括测试计划、测试设计和测试开发)，测试执行阶段(包括测试执行和测试评估)。

图1-9描述的软件测试流程H模型说明了软件开发某个阶段对应的测试活动，其他流程可以是任意的开发流程，如设计、编码等，也可以是非开发流程，如质量保证等，因此只要

测试条件成熟，准备工作就绪，就可以展开测试执行活动。在软件生命周期中的各个阶段都有这样独立的测试流程，如单元测试、集成测试等，这些流程与其他流程是独立的。H 模型体现了测试流程的独立性、完整性和过程管理的重要性，体现了“尽早测试和不断测试”的原则。

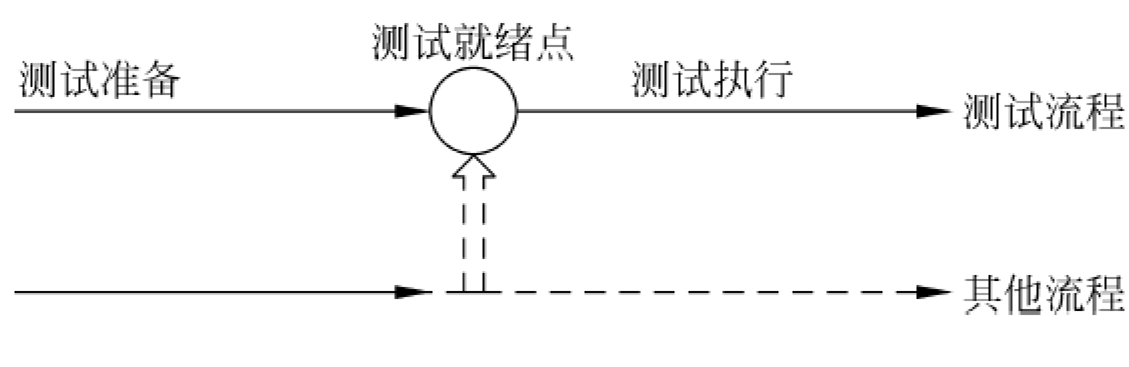


图 1-9 H 模型示意图

(4) X 模型。Marick 认为 V 模型无法引导项目的整个过程，从另外一个角度提出 X 模型弥补 V 模型的缺陷(W 模型从开发与测试的对应的角度进行改进)。图 1-10 给出了 X 模型的示意图，左半部分是针对单独程序片段进行的相互分离的编码和测试，经过多次交接，集成为可以执行的程序(右上半部分)。这些可执行的程序需要进行测试，已通过集成测试的成品可封版提交给用户，或者作为更大规模内集成的一部分。

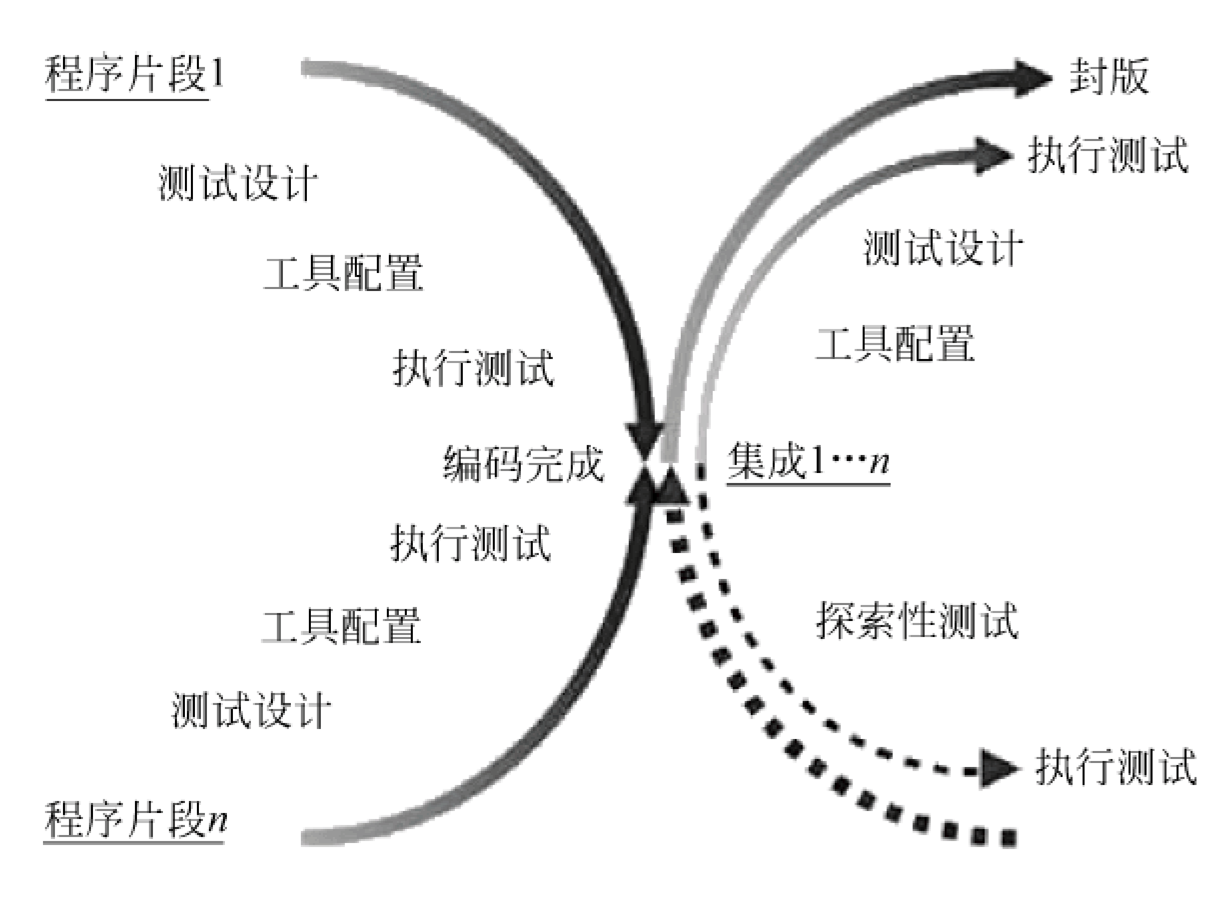


图 1-10 X 模型示意图

特别指出的是，X 模型提出了探索性测试，即无事先计划，只是随便测试一下，这样有助于有经验的测试人员在计划外发现更多软件缺陷。

(5) 测试成熟度模型。1996 年，人们参照 CMM 开发了软件测试成熟度模型 TMM，它包括 5 个等级，每个等级列出了一系列建议做法，企业可通过这些等级来评价自身的测试能力，以便进一步改进软件测试过程，促使软件测试向更强的专业化方向发展。

习题 1.8

1. 简单描述软件测试的信息流模型，指出在该模型中最关键的因素是什么？
2. 软件开发有哪些模型，软件测试分别处于什么位置？
3. 软件测试过程有哪些模型？这些模型各有什么特点？

1.9 软件测试的职业

几乎所有的职业都有一定的门槛，所谓门槛，就是其特殊的职业技能和基本素养，一个职业的重要程度不仅取决于社会需求，也取决于门槛的高低。例如，飞行员是一个非常重要的职业，过硬的身体和心理素质以及熟练的飞行驾驶技能要求形成了很高的职业门槛，使得能够从事这个职业的人员很少，培养一个成熟的飞行员的成本是非常昂贵的。

软件测试从业人员不仅需要系统地掌握软件测试的基本知识，还要在实践中不断摸索，将理论联系实际，善于不断尝试新的测试方法和测试工具，测试新软件，探索新问题，积累新经验。软件测试既是一门科学，也是一门艺术，测试员丰富的经验和敏锐的洞察力往往是成功的关键。一个测试人员，如果他掌握的软件测试方法越多，会用的测试工具越多，测试过的软件越多，碰到过的问题越多，那么他在新的测试任务到来的时候，成功的可能性就越大，这样的测试人员价值也就越大。

从最广泛的意义上来讲，测试无处不在，因为各行各业都会有各自的产品，在他们的产品出厂前，都需要进行测试，以保证产品质量合格。同时我们生活的每个角落，例如，到商场买东西，要看看产品质量；跟别人交往，要测试对方是否友好等；特别是在社会生活中我们也一直在接受来自学校、单位或其他个人的测试。正因为测试无处不在，所以可以说人人都是测试员，每个人都在利用自己独立的视角或者说独具慧眼在生活中进行着辨识和选择。软件作为一种特殊的产品，软件测试是一种特殊的测试，因此有着非常广泛的从业人员基础，软件测试从业人员可以有广泛的背景，从对专业知识一无所知的门外汉到百分之百的领域专家，当然更多的测试员是介于两者之间。但在这个行业中要做到一个成熟的软件测试专业人员，还是非常困难的，需要厚实的专业基础，不断地学习、实践和积累。

软件测试的专业基础涉及数学、计算机科学、软件工程、计算机工程以及信息技术等非常广泛的领域，而不是仅仅学好软件测试这门课程就可以了，而且如果没有这些专业基础，软件测试课程也是学不好的，关于这一点，从软件测试课程中广泛的知识体也可以看出。一般来讲，都是计算机或相关专业的毕业生毕业后根据兴趣和需要，经过一定的培训或学习后，开始从事软件测试职业。

人们总是喜欢根据从业人员的能力和成熟度将一个职业分成几个等级，在软件测试领域也可以将从业人员分成5等级(也可以更多，这里只给出5个等级作为参考)：1级称之为用户级测试人员，这是软件测试人员的最低级别，没有受过软件测试的专门训练，直接从用户角度通过使用软件来发现问题；2级是软件测试操作员，受过软件测试的专门训练，不仅可以从用户角度进行用户级测试，还可以完成上级下达的具体的测试任务；3级是软件测试技术员，能够非常熟悉地驾驭某些软件测试专门技术，设计一些高级的测试用例，进行高效的软件测试，可以将一些测试任务下达给2级测试操作员，并具有很多软件测试成功的经验；4级是软件测试系统工程师，系统掌握所有的软件测试方法，具有计算机等相关专业知识，并曾是成熟的软件开发人员，具有充分的软件测试经历；5级是软件测试总工程师，既是软件测试系统工程师，具有丰富的软件测试经验，又能做项目经理，负责管理整个软件测试项目。

习题 1.9

1. 作为一种职业，你认为软件测试员应该具备哪些专业基础？

2. 实践经验是人们称为一名合格专业人士的必备条件，你认为软件测试人员的实践经验包括哪些内容？

1.10 软件测试的学术研究

软件测试领域存在着很多非常困难的科学问题，例如，对于一个特定的软件，如何选择一组有效的测试方法，对之进行科学的测试？如何从庞大的可用测试用例空间中选择少量的测试用例对该软件进行有效的测试？软件测试什么时候可以停止？等等。正是这些问题推动着软件测试领域学术研究的繁荣，很多研究者试图从自己的视角和基础出发给出新的解决方案。

2000 年，M. J. Horrold 对软件测试领域进行总结，提出软件测试领域的主要问题是探索各种新的软件测试方法和过程，开发相应的工具，并进行实证研究。2007 年，Antonia Bertolino 又重新对该领域进行总结，她首先肯定人们在可靠性测试、测试过程、协议测试、测试充分性准则及相互比较、构件测试及面向对象软件测试方面取得的成绩。然后指出了人们在测试输入生成、测试预期输出、测试成本及有效性、基于假设的测试、针对不同计算方式软件的测试、功能性和非功能性测试等领域的挑战，提出了建立统一的测试理论、基于测试的建模、测试自动化和效率最大化测试工程等 4 项研究目标。

本书将软件测试的研究划分为两大类：一类是图 1-11 中上面部分，研究针对各类具体软件的测试方法；另一类是下面部分，可称为软件测试的基本方法。下面的方法体系可为上面具体软件的测试服务，同时具体软件测试也给基本测试方法提出了新的要求。

				针对不同开发方式和应用场景的软件测试方法													
				面向对象软件测试	面向方面软件测试	面向服务软件测试	基于构件软件测试	嵌入式软件测试	普适环境软件测试	云计算软件测试	Web应用软件测试	网构软件测试	其他新型软件测试				
软件测试的基本方法	静态测试	桌面检查	代码审查	代码走查	右下方三行属于动态测试			针对软件不同特性和方面的软件测试方法									
								负载测试	压力测试	性能测试	安全性测试	安装测试	可用性测试	稳定性测试	配置测试	文档测试	兼容性测试
	动态测试	黑盒测试	等价类划分	边界值分析	因果图分析	错误猜测	状态转换测试	针对不同开发阶段的软件测试方法									
								单元测试	集成测试	系统测试	验收测试	回归测试	验证测试	确认测试	α测试	β测试	γ测试
								不同特殊的软件测试方法									
		白盒测试	语句覆盖	判定覆盖	条件覆盖	路径覆盖	条件组合覆盖	组合测试	蜕变测试	变异测试	演化测试	模糊测试	基于性质的测试	基于故障的测试	基于模型的测试	统计测试	逻辑测试

图 1-11 软件测试的方法体系

图 1-11 下面部分的软件测试基本方法中的每一种方法都有其独特的测试目标，用以解决不同的测试问题，采用不同的测试用例生成方法，从而具有不同的特点。例如，组合测试目标是检测待测软件系统中各种因素相互作用引发的故障，采用组合设计方法生成测试用例，可以有效地发现交互性错误。边界值分析根据人们实践经验：边界点往往有较大可能引发错误，因此在设计测试用例时，充分采用边界值。每一种方法在使用时，其效果往往因人而异，原因就是每一种方法都值得研究，我们需要对其中每一种方法都要进行认真研究，探索各种方法的具体使用步骤、工具支持、使用成本、存在的风险等问题。

目前关于某种特殊的测试方法的研究非常多，例如蜕变测试、变异测试、组合测试、演化测试等等，这些研究一方面研究和尝试如何充分发挥特定的测试方法的作用，另一方面，通过一些具体的应用，收集该方法有效性的证据，并在实践中发现和解决一些新的问题。例如，在软件发生修改后，研究如何以最小的测试用例集、最少成本的测试保证修改的正确性以及修改未带来任何副作用；怎样定量评估各种测试充分性准则的有效性等问题。其实，测试用例集的最小化和测试有效性评估也是所有测试方法的共性问题，除此之外，还有测试目标及测试用例生成问题；测试用例的优化，即寻找一个最好的测试用例执行顺序，使之达到最好测试效果；还有在用不同的测试方法时，如果发现故障，如何进行故障定位问题；不同测试方法的测试充分性问题等。

除了对单个测试方法进行系统研究，还需要从软件测试不同层面进行系统的交叉结合和比较研究，例如，针对各种测试覆盖标准的比较研究，软件开发过程中不同阶段的测试间关系研究，面向各种具体开发方法和应用环境的软件测试方法的比较研究，软件的各种特性测试，以及各种测试技术之间的比较研究等，目标是探索并建立软件测试研究和实践的多维度和系统化的科学体系。例如，人们已经对随机测试和自适应随机测试、随机测试和组合测试等进行比较研究。

习题 1.10

1. 在软件测试领域，最根本的问题在哪里？为了解决这个问题，人们通常采取哪些方法？

2. 软件测试方法的理论与实证研究都很重要，各种测试方法的交叉结合的意义是什么？

1.11 软件测试的工具

工具是人们认识世界、改造世界能力的延伸，例如，有了手机，人们可以和千里之外的朋友说话；有了飞机，人们可以日行万里。软件测试工具可以将原来人们手工完成的工作，利用工具辅助完成，可以极大减轻测试人员的工作量，提高工作效率。例如，有些测试数据生成工具，可以在短时间内产生大量的数据，而不需要测试人员一条条地添加数据。有些工具在提高工作效率方面的作用并不明显，但可以起到让测试更规范。例如，测试管理工具或缺陷跟踪工具，可以让测试的流程更加规范，让测试有条不紊地进行，测试报告更加规范，数据记录更加完整，统计分析更加准确等。

软件测试工具大致包括以下几种：

(1) 测试管理工具。用于管理测试的整个工作过程以及过程中产生的各种相关文档、数据、记录和报告等。常用的管理工具有 TestDirector 和 TestManager 等。

(2) 自动化功能测试工具。用于自动化执行功能测试脚本，一般采用基于录制回放的机制，通过录制操作过程，产生基本的测试脚本，然后在脚本编辑器中进一步完善测试脚本，再通过回放脚本的方式执行测试脚本的步骤，从而实现功能测试的自动化。常用的测试自动化工具有 QTP(Quick Test Professional)、Rational Robot、TestCoplete 等。

(3) 性能测试工具。用于性能测试过程中的通信协议模拟、并发用户模拟及性能参数监控等方面的测试工具，如 LoadRunner、SilkPerformer 等。

(4) 单元测试工具。用于单元测试的测试框架，这些测试工具提供单元测试的一些接口，管理单元测试的执行，例如 XUnit 系列、MSTest 等。

(5) 白盒测试工具。用于测试程序内部逻辑和错误的测试工具，又可以细分为代码标准检查工具、代码效率检查工具和内存泄露检查工具等，如 TEST、AQTime 和 BundsChecker 等。

(6) 测试用例设计工具。用于辅助测试用例的设计或测试数据生成的工具，一般常用的有 CTE XL、AETG 和 PICT 等。

测试工具还可以根据收费方式分为商业测试工具、开源测试工具和免费测试工具。

习题 1.11

1. 软件测试工具大致可以分为几种？
2. 学习使用一种软件测试工具。

1.12 软件测试的管理

随着软件规模和复杂性的日益增长，软件故障也越来越多，作为软件质量保证的重要手段，软件测试的规模和难度也越来越大，需要的测试人员也越来越多。对测试过程、测试人员及组织、测试文档、测试环境、测试质量等都需要进行全面管理，才能有效进行测试。

软件测试项目是在一定的组织机构内，利用有限的人力和财力资源，在特定的环境和要求下，对特定软件完成指定测试目标的测试任务，测试任务要满足一定质量、数量和技术指标要求。

软件测试管理就是以测试项目为管理对象，通过一个临时性的专门的测试组织，运用专门的软件测试知识、技能、工具和方法，对测试项目进行计划、组织、执行和控制，并在时间成本、软件测试质量等方面进行分析和管理活动。软件测试管理贯穿整个测试项目的生命周期，是对测试项目的全过程进行管理。

软件工程项目越大，越复杂，管理工作量占软件研制工作量的比例也越大。管理的基本目标是以最小代价达到对工程项目预期的要求，基本任务是保证恰当地确定软件测试需求

和圆满地实现要求。因此,软件测试管理的关键在于,对软件测试全过程控制,对软件质量的全方位管理,建立多层次的软件开发、管理体系。

王铁辰等为软件测试管理构建了一个全方位、全过程、多层次的三维模型:

(1) 时间维。全过程管理,对软件测试项目的全过程进行控制,具体包括:测试计划管理、测试设计管理、测试执行管理、测试结果管理等。

(2) 空间维。全方位管理,对与软件质量有关的关键因素实施全方位管理,具体包括缺陷管理、文档管理、配置管理、评审管理、质量管理和回归管理等。

(3) 组织维。人员管理,构建从测试人员、测试小组到测试机构的多层次的组织管理模式。

习题 1.12

1. 软件测试的管理具体有哪些内容?
2. 如何理解软件测试管理的全方位、全过程和多层次。
3. 软件测试管理具有什么作用?

1.13　软件测试的过去、现在和未来

像计算机领域的其他学科一样,软件测试经历了从无到有、从简单到复杂、从低级到高级的发展过程。了解软件测试的过去和现状,有利于人们积极投入到软件测试的未来发展中去。

有了计算机软件,就可能存在问题;有了问题,就需要测试,因此可以认为软件测试是与计算机软件一起诞生的,即有了软件就有了软件测试。软件测试的发展是受计算机软件发展推动的,也反映了软件业的发展过程,根据软件业发展的几个历史时期可以将软件测试划分为以下几个阶段:

(1) 测试的初级阶段(1950 年左右)——面向调试(Debuging Oriented)。在这个时期,计算机技术发展初期,软件规模都很小,复杂度相对较低,软件错误大部分在开发人员的调试阶段就发现解决了,在这个阶段测试与调试是等同的,都是由开发人员自己完成。

(2) 软件测试与调试的分离(1960 年左右)——面向演示(Demonstration Oriented)。直到 1957 年,软件测试才开始与调试区别开来,有人认为"为了让我们看到产品在工作,就得将测试工作往后推一点",因此测试的目的就是"确信自己的产品能工作"。这个阶段的测试主要还是在程序编写后进行的,通过一些错误猜测和经验推断等方法验证软件的正确性,由于缺乏测试理论和方法的指导,测试的力度、时间都非常有限,软件交付后一般都存在大量问题。

(3) 测试的发展阶段(1970 年左右)——面向破坏(Destruction Oriented)。由于软件危机愈演愈烈,人们开始提出了软件开发的工程化思想,在软件工程的思想影响下,软件测试得到发展,软件测试的地位也得到了确认。人们开始给软件测试下定义,例如 1973 年,Bill Hetzel 博士给出了软件测试的第一个定义:"软件测试就是对程序能够按预期的要求运行

建立起一种信心”(1983 年修订为定义 2)。Myers 的定义 1“测试是以发现错误为目的而运行程序或系统的执行过程”也是这个时期给出的。

(4) 软件测试的日渐成熟阶段(1980 年左右)——面向评价(Evaluation Oriented)。软件行业的飞速发展,软件规模越来越大,复杂度越来越高,人们对软件质量越来越重视,软件测试的理论和技术都得到了快速的发展,人们开始把软件测试作为软件质量保证的重要手段,用于评价软件的质量。一种重要的标志是 IEEE 给软件测试下了定义,并发布了软件测试的国际标准——《软件测试文档 IEEE 标准》(IEEE 829)。人们已经认识到,软件测试作为检查软件系统是否满足用户需求的活动,不是一次性在开发后期完成的,而是伴随着整个开发流程;软件测试需要专门的方法和手段,需要专门的人才和专家来实施,应该成为一个专业。

(5) 软件测试的成熟阶段(1990 年后)——面向预防(Prevention Oriented)。随着软件工程的百花齐放,出现了面向对象方法、敏捷开发等各种软件开发的新模式,由此使得人们对软件测试进行了新的思考,人们渐渐倾向于将软件测试与软件开发进行融合,开发人员担任部分测试责任,测试人员参与到代码开发中去。软件开发与测试的界限在“测试驱动的开发模式(TDD)”中变得模糊起来。美国卡耐基梅隆大学软件工程研究所推出的能力成熟度模型(CMM)将软件测试整合在软件过程之中,作为衡量软件开发能力成熟度的一个重要指标,此阶段软件测试的特征是重在预防。

(6) 软件测试职业、教育和研究逐渐兴起(2000 年后)。出现了大批以软件测试为职业的从业人员和软件评测机构,在国外成熟的软件企业内,软件测试人员和开发人员的比例超过 1∶1,并成立了专门的测试部门;对成熟的软件测试人才的期盼已经成为所有软件企业最重要的需求。同时软件测试教育和研究成为软件工程领域一个非常重要的热点,几乎所有拥有计算机系的高校都开设软件测试课程,软件测试教材成批涌现,出现了很多软件测试学术研究组织和国际会议。每年都有大量的软件测试研究论文发表,不少专业公司开发了各种自动化测试工具,并形成了较大的相关产业。

软件测试领域在最近的 10 多年里发展迅速,前面已经从软件测试的定义、课程、软件测试职业和软件测试科学等多个层面试图对该领域进行一个系统的梳理,以期抛砖引玉,为软件测试领域的产学研紧密结合和更好更快发展提供新的系统化基础。通过系统地总结软件测试课程知识体,为软件测试课程的发展和教学改革提供新的基础和指导;通过探讨软件测试职业的职业技能和素养,为软件测试从业人员自身发展和提高提供参考;通过系统探讨目前软件测试研究领域的科学问题,可以为该领域科学工作者从事更加有效的研究工作提供新的思路。

不同的产品有不同的测试方法,所有这些测试方法全体构成一个大测试领域,软件是一种产品,软件测试是大测试领域中的一员(如图 1-12 所示)。软件测试领域的发展已经并将继续从大测试领域获得营养,即从其他产品测试方法中获得启发或借鉴,例如,从人才选拔测试过程中可以学习到像给学生考试一样考查软件;同时人们在软件测试领域本身取得的进步反过来也将推动大测试领域的进步。

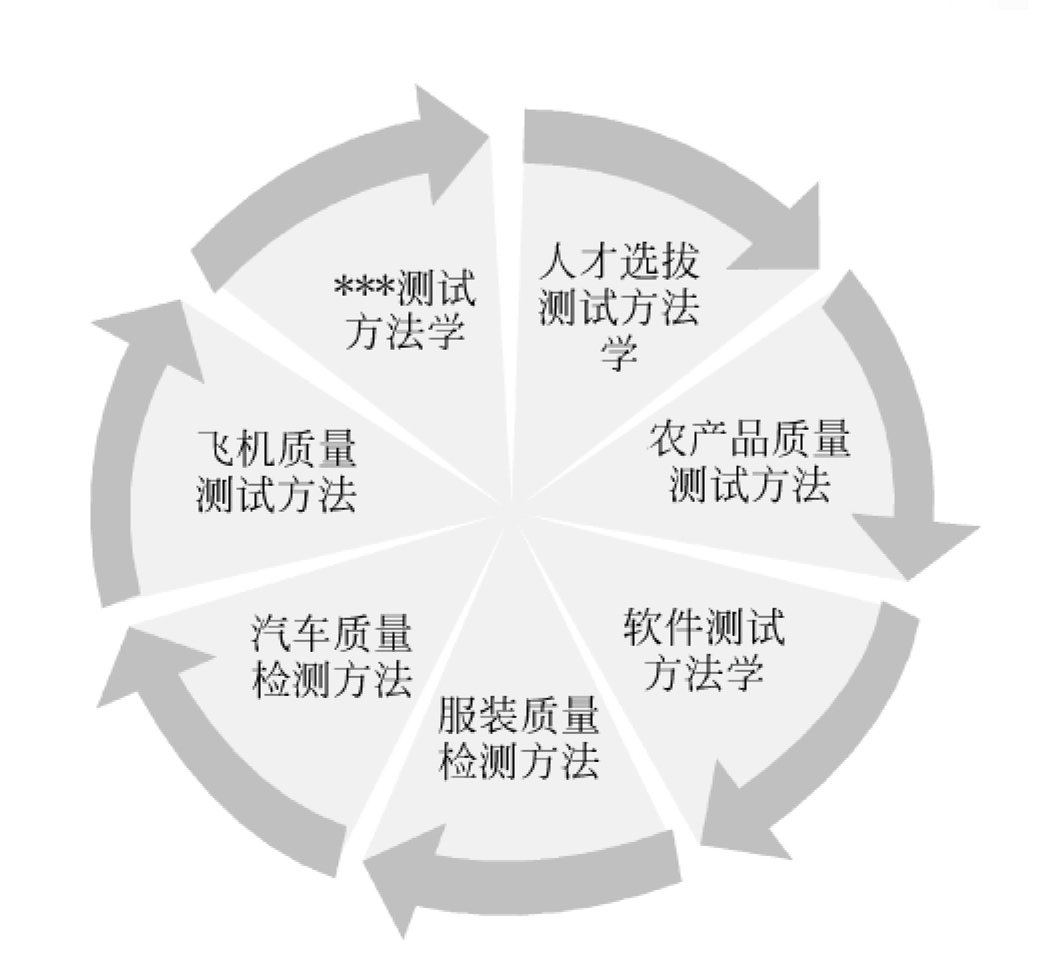


图 1-12　大测试领域中各类产品测试方法体系之间可以相互借鉴

习题 1.13

1. 简述软件测试发展的 6 个发展阶段。
2. 软件测试与其他领域产品测试方法应该如何相互促进发展？

第2章 白盒测试和黑盒测试

2.1 白盒测试(White Box Testing)

1. 概念

白盒测试对软件代码等过程性描述进行细致的检查,这一方法把测试对象看成一个打开的盒子(如图 2-1 所示),允许软件测试员利用程序内部的逻辑结构及有关信息,设计或选择测试用例,对程序所有逻辑路径和条件分支等结构成分进行测试。通过在不同点检查程序的状态,验证实际的状态是否与预期的状态一致,故又称结构测试。

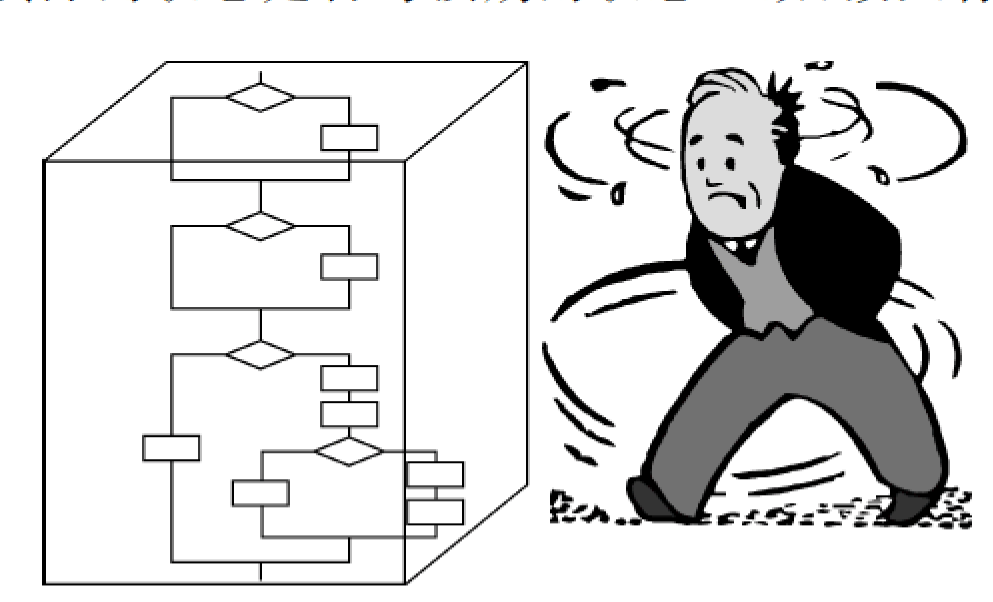


图 2-1　白盒测试示意图

2. 目标

白盒测试分为静态和动态两种。静态白盒测试包括代码审查、走查等,它不执行软件。程序的静态测试可以转变程序员的一些错误观点,例如"程序是写给计算机读的","只要在机器上可以执行就可以了"等。程序不仅要求机器可读和可执行,还要求对人具有良好的可读性,静态的白盒测试通过专家对程序员的工作进行评审和分析,其目的是:一方面要求程序员用更好的结构、更规范的方法工作,保证程序的可读性和可维护性;另一方面尽早发现程序中存在的问题。

动态白盒测试是运行待测试软件,检查其结构元素是否被完全覆盖等。使用动态白盒测试,主要对模块进行如下的测试:

(1) 对程序模块的所有独立的执行路径至少执行一次。

(2) 对所有的逻辑判定,取值为"真"与取值为"假"的两种情况都能至少测试一次。

(3) 在循环的边界与运行界限内执行循环体。

（4）测试内部数据结构的有效性。

3．原理

白盒测试必不可少，主要因为人们在设计和实现程序代码时可能存在以下问题：

（1）人们可能直觉上认为某些逻辑路径不可能被执行，但程序的逻辑流有时和直觉不一样，即人们关于控制流和数据流的一些无意识的假设可能导致设计错误，路径测试可以有效发现这类错误。

（2）人们在设计和编写软件代码时，一些常用的主流功能不大会出错，相反在设计和实现主流之外的功能、条件和控制时会出现各种疏忽和大意。程序员能很好地理解程序中的关键功能，而对一些非关键部分可能会投入不足。这就出现所谓"程序路径执行概率与错误存在概率成反比"的现象。

（3）程序在编码过程中，笔误、误解等各种类型的失误是不可避免的，虽然语法检查机制可以发现很多错误，但有些错误只能通过测试才能发现，而这些失误出现在每条逻辑路径上的几率都是一样的，因此细致的白盒测试非常有必要。

（4）功能性测试只能检查程序功能是否被正确实现，如果程序实现了没有被描述的行为，功能性测试就无法发现。例如，程序中存在一些安全漏洞，甚至是病毒，这些都会给软件带来隐患。

白盒测试与调试的关系如下：

白盒测试是为了发现错误，调试是根据发现错误的测试记录报告，对错误进行定位和修改，最后消除错误的过程；白盒测试可以由独立的测试人员进行，但调试一般只由程序员本人完成；白盒测试是按照一定计划进行的，需要进行精心设计，调试则依赖于程序员的经验和智慧进行推理。两者具有紧密的联系，它们都是对软件代码等过程性描述进行细致的检查。白盒测试需要调试将发现的错误消除，调试需要白盒测试将发现的错误进行隔离，并确认错误已经被修正。

4．方法

白盒测试基于源程序的测试方法，根据是否执行源程序分为静态白盒测试和动态白盒测试。静态白盒测试是指按照一定步骤直接检查源代码来发现错误，而不用生成测试用例并驱动被测试程序运行来发现错误。静态白盒测试包括代码审查、代码走查、桌面检查、静态结构分析法和代码质量度量法等方法。动态白盒测试需要基于对程序代码内部逻辑的了解，按照一定步骤生成测试用例并驱动程序运行来发现错误，其实现主要基于以下各种覆盖准则：

- 语句覆盖；
- 判定覆盖；
- 条件覆盖；
- 判定/条件覆盖；
- 条件组合覆盖；

- 路径覆盖；
- LCSAJ 覆盖；
- 定义/引用对。

白盒测试主要有以下步骤：根据软件高层设计、详细设计和源程序等画出程序图，设计并执行测试用例，分析覆盖标准和判定测试结果。图 2-2 描述了白盒测试过程，包括静态和动态两个方面。在实际测试过程中，往往先进行静态白盒测试，然后进行动态白盒测试；或者将这两个方面结合在一起使用。例如，在静态白盒测试过程中设计动态测试用例，或者在设计动态测试用例的过程中进行静态白盒测试，这样可以设计出更具针对性的白盒测试用例。

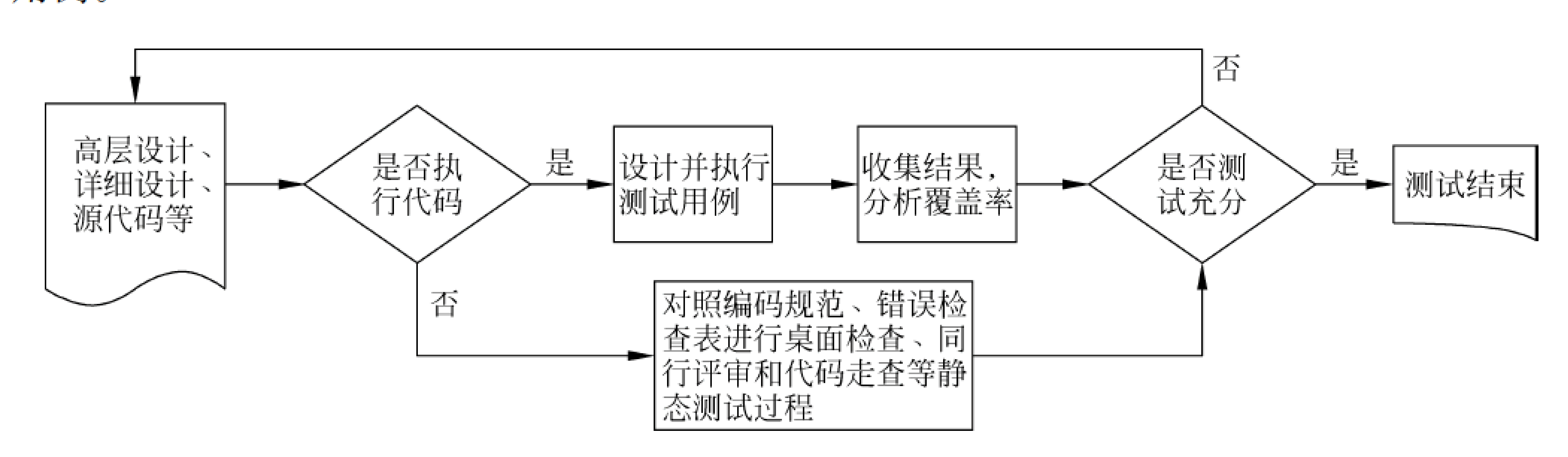


图 2-2 静态/动态白盒测试过程

5. 实例

如图 2-3 所示的程序流图，它对应了一个 100 行源代码的 C 语言程序，其中包括一个执行不超过 20 次的循环，那么它所包含的不同路径高达 5^{20} ($\approx 10^{14}$) 条。若要对它进行穷举测试，即设计测试用例并覆盖所有的路径，假设有一个测试程序，对每条路径的测试需 1ms，那么要完成测试约需 3024 年。根据这个简单例子，我们可以得出以下两点：

(1) 进行任何穷举测试都是一场灾难，因此，可行的策略是在一定的开发周期和某种经济条件下，通过有限的测试尽可能多地发现错误。

(2) 测试只能发现错误，而在未发现错误的情况下，不能保证程序没有错误。

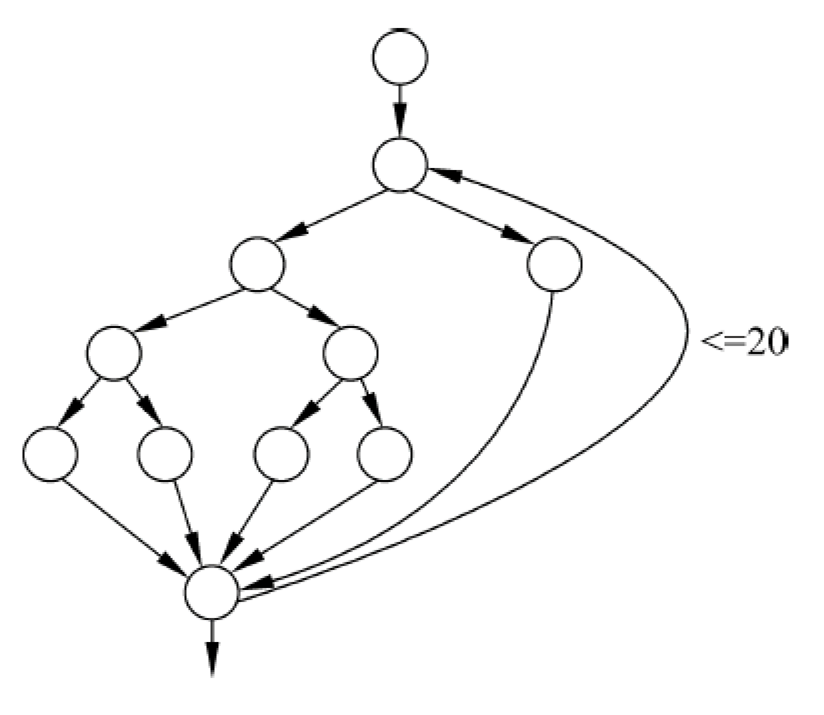


图 2-3 程序流图

6. 优缺点

白盒测试是在具备实现知识的基础上的静态或动态测试，例如，代码评审、代码走查、要求每行代码至少执行一次或每个函数被单独执行一次等。白盒测试的主要目标是关注内部程序结构。关注的程序结构包括程序语句和分支、各种类型程序路径、程序内部逻辑和数据结构、程序内部行为和状态，以此为基础发现内部程序错误。在黑盒测试基础上增加白盒测试，可以有效提高软件质量。

白盒测试，特别是静态白盒测试的优点是不仅可以尽早发现问题，而且有助于优化代

码。例如,通过分析源代码,确定是否与编码标准一致;删除多余的代码行,减少隐藏缺陷;发现哪种输入数据有利于提高测试效率等。动态白盒测试针对源代码进行代码覆盖、分支覆盖和路径覆盖等多方面测试求证,形成对软件的健壮测试。

白盒测试的不足之处是需要了解代码和内部结构,因此有较高的门槛,需要对目标系统、测试工具、编码语言和建模等具备丰富经验的测试人员。特别是对于一些大型软件,必须使用一些特殊的工具,如源代码分析工具、调试器和故障植入工具等进行辅助。

习题 2.1

1. 白盒测试可以检查出哪些问题?它有哪些优点和不足?
2. 白盒测试包含哪些不同级别的测试方法?

2.1.1　静态白盒测试(Static White Box Testing)

1. 概念

静态白盒测试是一些评审过程,可以是几个程序员之间一个简单的会议,对软件的设计和编码进行详细和严格的审查等。这些评审过程有以下4个特征:

(1) 发现问题。针对设计和代码,检查错误和遗漏,所有的批评都是对事不对人。

(2) 遵循一定的规则。例如,指定审查的代码量、会议时间和每个参会人员的责任等。

(3) 准备工作。评审会效果取决于参会人员的会前准备工作,每个人都要为会议作出贡献。很多问题都是在会前准备发现的,而不是在会议上。

(4) 会议报告。评审团要求就发现的问题写一个书面报告,内容包括发现了多少问题、问题在哪里等。

评审过程需要一个严格成熟的过程,不能停留在“聚到一起,看一看代码”的初级阶段,否则,起不到检查和发现错误的目的,而只是浪费时间。当然,评审过程不可能发现所有的问题,该过程无法发现的问题将通过动态执行阶段等后续方法发现。

静态白盒测试通过各种评审,除了可以发现问题外,还有以下作用:

(1) 交流沟通的作用。通过评审会,程序员之间可以相互学习;项目经理可以了解项目进度;黑盒测试人员可能会得到一些有用信息,有利于进行黑盒测试。

(2) 提高编码和文档质量。程序的代码和相关的文档被同行和专家仔细阅读和评审,这对程序员是一个很大的压力,这使得程序员在编码及文档时会非常认真。

(3) 建立团队友谊。评审会使得大家相互了解对方的工作技能和难度,互相理解和尊重。

(4) 寻求解决方案。对于一些棘手的问题,评审会的讨论有可能提供部分解决思路。

2. 方法

(1) 同行评审(Peer Review)。有时也称好友评审,不是很正式,相当于“你把你的程序拿给我看看,我把我的程序拿给你看看”,通常由软件设计人员和编码人员加上一两个其他程序员或测试人员组成。要提高同行评审的效果,必须着重体现一般评审过程的4个特征。不管怎样,聚到一起讨论程序代码还是有助于发现问题的。

（2）代码审查(Code Inspections)。代码审查是最正式的一种评审，编写受评审代码的程序员不能参与评审，所以要求每个参与人员都要受过专门的训练，能够学习和理解代码和相关的材料。每个评审人员可以从用户、测试人员和产品支持人员等不同角度对代码进行评审，从而可以从不同方面发现错误。要从其中指定一个记录员和一名协调员负责整个审查过程的有效运行。审查会议结束之后，评审人员与协调员就发现的问题编写一个报告，指出解决相关问题的必要性工作，然后交给程序员进行整改；协调员对相关修改进行查证，并根据问题的大小决定是否进行下一轮审查。代码审查可以有效发现程序中潜藏的错误，很多公司和开发团队都使用这种方法。

（3）代码走查(Code Walkthroughs)。代码走查更正式一点，由程序员和测试人员组成的5人左右的审查小组审查程序员自己的程序。审查小组成员应该提早收到程序拷贝，认真阅读，写好评语和关心的问题。审查小组中至少要有一个资深的程序员。

程序员通过一行一行、一个功能一个功能地阅读代码，解释代码是做什么的以及原因，评审组听取汇报，并就可疑的地方进行提问。参与代码走查的人员比同行评审更多一点，因此，更要在发现错误、会议规则、提前准备和会议总结等环节下工夫。程序员最后将走查过程中发现的问题及如何解决这些问题的初步计划编写一个报告。

（4）桌面检查(Desk Examination)。桌面检查可视为由单人进行的代码检查或代码走查，由一个人阅读程序，对照错误列表检查程序，对程序推演测试数据。

桌面检查一般效率比较低，主要因为桌面检查基本上没有约束，它违反了程序员不能检查自己程序的原则。改进的桌面检查由程序员交互检查各自的程序，但即使是这样，其效果仍然不如代码审查或代码走查。因为在代码审查和代码走查小组中存在着相互促进的效应，小组会议培养了良性竞争的气氛，人们喜欢通过发现问题来展示自己的能力。在桌面检查过程中，由于没有机会向其他人展示，因此缺乏这种相互促进的效果。桌面检查好过没有检查，但其效果远远逊色于代码审查和代码走查。

3. 编码标准和指南

在正式评审中，代码审查主要发现一些遗漏和问题。一类问题是代码写得没有问题，但程序不一定能按照代码正常工作，这些问题必须通过资深的程序员和测试人员仔细分析代码才能发现；另一类问题是程序可以正常工作，但代码的编写不符合一定的标准规范和指南。标准规范是必须要参照执行的，而指南只是建议。程序编写的标准规范和指南是非常重要的，主要有以下原因：

（1）可靠性要求。按照一定的标准规范和指南编写的程序将具有更好的安全性和可靠性。

（2）可读性和可维护性要求。按照一定的标准规范和指南编写的程序更容易阅读、理解和修改维护。

（3）可移植性要求。代码要求能在不同硬件上运行，在不同编译器上编译。如果它按照一定规范，在不同平台间转移就可以非常容易。

无论标准，还是松散的团队内部指南，最关键的是开发队伍要有一些标准和指南，并通过正式的评审确保遵照标准执行。

习题 2.1.1

1. 静态白盒测试有哪些形式？它们各有哪些利弊？
2. 查找一些资料，给出一些编码标准和指南的实例。

2.1.2 语句覆盖测试(Statement Testing)

语句是构成软件程序的元素，软件测试应该测试每条语句。语句覆盖就是以源代码为基础，设计测试用例，执行其中每条可执行语句。每个测试用例必须指明 3 个方面的内容，即测试用例的测试输入、该测试用例执行的语句、测试用例的预期输出。

在语句覆盖中，覆盖对象是每条可执行语句。可执行语句包括赋值语句、循环和选择语句、过程和函数调用语句、带初始化的变量声明语句和堆中的可变存储的动态分配语句等。普通的变量申明语句被认为是不可执行语句。一般地，一条原子语句是一个要么执行，要么不执行的语句。例如 If a Then b，就不是一条原子语句，因为其中的 b 可能执行，也有可能不执行，这取决于 a 的取值。

语句覆盖率计算公式如下：

$$\text{语句覆盖率} = \frac{\text{已经执行的语句数目}}{\text{程序中所有可执行语句数目}} \times 100\%$$

例如，考虑 C 语言代码：

```
a;
if (b) {
      c;
      }
d;
```

设计一个测试用例，执行 b 的真值分支，就可以实现语句覆盖，但如果执行 b 的假值分支，则无法覆盖语句 c。

习题 2.1.2

结合自己的编程实践，给出一个通过语句覆盖测试就可以查出该程序中存在错误的例子程序，并说明语句覆盖测试的重要性。

2.1.3 分支/决策覆盖测试(Branch/Decision Testing)

分支/决策覆盖测试要求从源代码中找出所有的分支或决策点，设计相应的测试用例，执行所有的分支或每个决策点的两种取值。每个测试用例要指出 3 个方面的内容，即测试用例的测试输入、该测试用例执行的分支或决策点的真假值、测试用例的预期输出。

$$\text{分支/决策覆盖率} = \frac{\text{已执行的分支/决策个数}}{\text{源代码中所有的分支/决策个数}} \times 100\%$$

例如，图 2-4 中的程序是在一个按照字典序排序的单词表中查找某个单词的二分查找方法，图 2-5 给出了该程序的控制流图和其中所有的分支，表 2-1 给出了分支覆盖的测试用例。

```
          int binsearch (char *word, struct key tab[], int n) {
                int cond;
                int low, high, mid;

B1              low = 0;
                high = n - 1;
B2              while (low <= high) {
B3                    mid = (low+high) / 2
                      if ((cond = strcmp(word, tab[mid].word)) < 0)
B4                          high = mid - 1;
B5                    else if (cond > 0)
B6                          low = mid + 1;
B7                    else
                            return mid;
B8              }
B9              return -1;
          }
```

图 2-4　字典序的单词表中二分查找

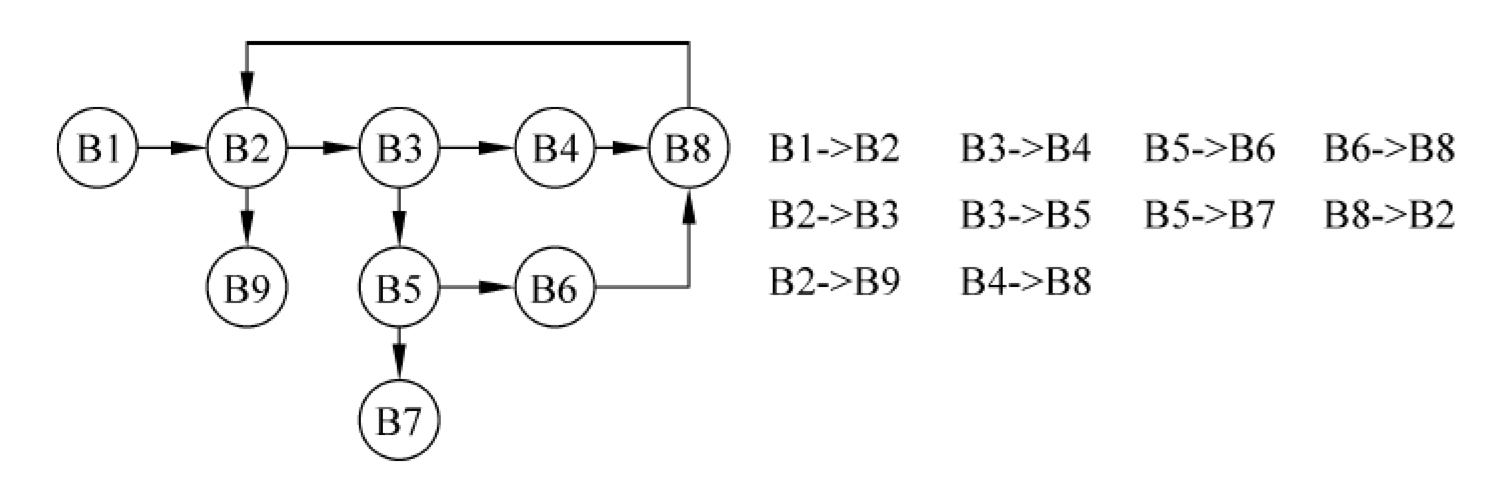


图 2-5　二分查找程序的控制流图及其中所有分支

表 2-1　二分查找程序的分支覆盖测试用例

测试用例	输　入			执行过的分支(加下划线)	预 期 输 出
	word	tab	n		
1	chas	'empty table'	0	B1 » B2 » B9	－1
2	chas	alf bert chas dick eddy fred geoff	7	B1 » B2 » B3 » B4 » B8 » B2 » B3 » B5 » B6 » B8 » B2 » B3 » B5 » B7	2

习题 2.1.3

分支覆盖可以做到语句覆盖,反之是否仍然成立?

2.1.4　数据流测试(Data Flow Testing)

数据流/控制流测试利用程序的数据流/控制流图进行测试,数据流/控制流图是将程序

中各个部分通过数据流/控制流连接起来而构建的程序的交互模型。在程序中变量的出现被分为两类,一类是定义,另一类是使用。如果一个变量被赋予新值,称该变量被定义;变量的使用可以分为两类:一种是变量出现在 while…do…或 if …then…else…等分支语句的谓词部分,这种使用称为谓词使用(P-use);另一种是变量出现在赋值语句的右边,或者出现在输出语句中,这类使用称为计算使用(C-use)。在变量定义和使用的每条路径都需要设计测试用例来执行,且每个测试用例应该包括测试输入、相关变量的定义和使用对的位置、执行的控制流子路径、预期输出。

在数据流/控制流测试中,覆盖对象是从变量的定义到使用的控制流子路径。为了覆盖率计算的需要,这里给出定义引用对(Definition-use Pair)的概念:一个变量的定义引用对是程序控制流图中一段连接该变量定义和使用的一条简单路径。所谓简单路径是指程序控制流图中没有不必要的(冗余)路径。以下给出覆盖准则的具体定义。

- 所有定义覆盖(All-definitions) 设计测试用例并执行所有变量定义点到某些使用点(计算使用或者谓词使用)的路径。
- 所有计算使用覆盖(All-C-uses) 设计测试用例并执行每个变量定义点到其每个计算使用点的路径。
- 所有谓词使用覆盖(All-P-uses) 设计测试用例并执行每个变量定义点到其每个谓词使用点的路径。
- 所有使用覆盖(All-uses) 设计测试用例并执行每个变量定义点到其每个使用点(计算使用或者谓词使用)的路径。
- 所有定义使用对覆盖(All-du-paths) 设计测试用例并执行所有变量定义点到每个使用点(计算使用或者谓词使用)的路径。

以下通过一个简单的例子说明各个概念。

例 2-1 图 2-6 列出的是一个求平方根的程序。

```
procedure Solve_Quadratic(A, B, C: in Float; Is_Complex: out -- 0
                          Boolean; R1, R2: out Float) is

-- Is_Complex is true if the roots are not real.
-- If the two roots are real, they are produced in R1, R2.

      Discrim : Float := B*B - 4.0*A*C;                     -- 1
      R1, R2: Float;                                        -- 2
   begin                                                    -- 3
      if Discrim < 0.0 then                                 -- 4
         Is_Complex := true;                                -- 5
      else                                                  -- 6
         Is_Complex := false;                               -- 7
      end if;                                               -- 8
      if not Is_Complex then                                -- 9
         R1 := (-B + Sqrt(Discrim))/ (2.0*A);               -- 10
         R2 := (-B - Sqrt(Discrim))/ (2.0*A);               -- 11
      end if;                                               -- 12
   end Solve_Quadratic;                                     -- 13
```

图 2-6 求平方根程序

表 2-2 中列出了这个程序中出现的各个变量,给出了各个变量定义的位置、计算使用和谓词使用的位置(代码行)。

表 2-2 求平方根程序中出现的变量及其分类

行	类别		
	definition	C-use	P-use
0	A,B,C		
1	Discrim	A,B,C	
2			
3			
4			Discrim
5	Is_Complex		
6			
7	Is_Complex		
8			
9			Is_Complex
10	R1	A,B,Discrim	
11	R2	A,B,Discrim	
12			
13			

表 2-3 给出了变量的定义使用对,其中 R1 和 R2 在该程序中无法给出定义使用对,因为它们分别只在一处(在第 10 行和第 11 行)被定义。

表 2-3 求平方根程序中变量的定义引用对

定义使用对(起始行→终止行)	变量	
	C-use	P-use
0--->1	A,B,C	
0--->10	A,B	
0--->11	A,B	
1--->4		Discrim
1--->10	Discrim	
1--->11	Discrim	
5--->9		Is_Complex
7--->9		Is_Complex

表 2-4 给出了实现 100%的所有定义覆盖(All-definition)所需要的测试用例,其中一个测试用例可以覆盖多个变量的定义。表 2-4 中测试用例 1、2、3 都是一样的。表 2-4～表 2-7 中的“unass.”是 unassignment 的缩写,表示未赋值。

表 2-4 所有定义覆盖(All-definition)测试用例

	All Definitions			输入			预期输出		
测试用例	变量	du-pair	subpath	A	B	C	Is_Complex	R1	R2
1	A,B,C	0-->1	0-1	1	1	1	T	unass.	unass.
2	Discrim	1-->4	1-4	1	1	1	T	unass.	unass.
3	Is_Complex	5-->9	5-9	1	1	1	T	unass.	unass.
4	Is_Complex	7-->9	7-9	1	2	1	F	−1	−1

表 2-5 给出了 100%的计算使用所需要的测试用例，表 2-6 给出了 100%的谓词使用所需要的测试用例，表 2-7 给出了 100%的所有使用覆盖所需要的测试用例。

表 2-5　所有计算使用覆盖(All-C-use)测试用例

	All-C-uses			输入			预 期 输 出		
测试用例	变量	du-pair	subpath	A	B	C	Is_Complex	R1	R2
1	A,B,C	0-->1	0-1	1	1	1	T	unass.	unass.
2	A,B	0-->10	0-1-4-7-9-10	1	2	1	F	−1	−1
3	A,B	0-->11	0-1-4-7-9-10-11	1	2	1	F	−1	−1
4	Discrim	1-->10	1-4-7-9-10	1	2	1	F	−1	−1
5	Discrim	1-->11	1-4-7-9-10-11	1	2	1	F	−1	−1

表 2-6　所有谓词使用覆盖(All-P-use)测试用例

	All-P-uses			输入			预 期 输 出		
测试用例	变量	du-pair	subpath	A	B	C	Is_Complex	R1	R2
1	Discrim	1-->4	1-4	1	1	1	T	unass.	unass.
2	Is_Complex	5-->9	5-9	1	1	1	T	unass.	unass.
3	Is_Complex	7-->9	7-9	1	2	1	F	−1	−1

表 2-7　所有使用覆盖(All-use)测试用例

	All-uses/All du-paths			输入			预 期 输 出		
测试用例	变量	du-pair	subpath	A	B	C	Is_Complex	R1	R2
1	A,B,C	0-->1	0-1	1	1	1	T	unass.	unass.
2	A,B	0-->10	0-1-4-7-9-10	1	2	1	F	−1	−1
3	A,B	0-->11	0-1-4-7-9-10-11	1	2	1	F	−1	−1
4	Discrim	1-->4	1-4	1	1	1	T	unass.	unass.
5	Discrim	1-->10	1-4-7-9-10	1	2	1	F	−1	−1
6	Discrim	1-->11	1-4-7-9-10-11	1	2	1	F	−1	−1
7	Is_Complex	5-->9	5-9	1	1	1	T	unass.	unass.
8	Is_Complex	7-->9	7-9	1	2	1	F	−1	−1

虽然所有定义使用对覆盖标准要求所有连接变量定义和使用的路径必须都要被覆盖，其实有些路径是不可行的，例如，0-1-4-5-9-10 和 1-4-5-9-10。

习题 2.1.4

数据流覆盖具有哪几个覆盖准则？它们之间的关系如何？

2.1.5　条件覆盖测试(Condition Coverage Testing)

用图 2-7 给出的简单例子说明各种条件测试的标准和相互之间的关系。

条件覆盖测试是指设计测试用例，让判定语句中的每个条件变量的真假值都能取到。

```
if A or (B and C) then
        do_something;
else
        do_something_else;
end if;
```

图 2-7　程序例子

表 2-8 给出了一个条件覆盖测试用例，注意到在这两个测试用例中，每个条件的真假值都出现过一次。条件覆盖测试不能保证分支或者决策覆盖，表 2-9 给出的一组条件覆盖测试用例就只能覆盖判定的真值部分，假值分支覆盖不了，因此有必要增加下一个覆盖标准，即分支条件覆盖。

表 2-8　条件覆盖测试用例集

测试用例	A	B	C
1	False	False	False
2	True	True	True

表 2-9　只覆盖真值分支的条件覆盖测试用例集

测试用例	A	B	C
1	True	False	False
2	False	True	True

习题 2.1.5

条件覆盖是否可以做到分支覆盖？

2.1.6　分支条件覆盖测试（Branch Condition Testing）

设计测试用例不仅让判定语句中的每个条件变量的真假值都取到，而且让判定语句取到真假值。实际上表 2-8 中两条测试用例就是一组分支条件覆盖测试用例，它们既保证每个变量的真假值都出现一次，又能保证判定语句分别取真值和假值。

分支条件覆盖既考虑了分支覆盖，又考虑了条件覆盖，是一种强于分支覆盖和条件覆盖的测试方法。但分支覆盖与条件覆盖之间不具有可比性，因为分支覆盖不一定就能条件覆盖，而反之，条件覆盖不一定能做到分支覆盖。例如，2.1.5 小节中的表 2-9 是一个条件覆盖测试用例集，但该测试用例集只能覆盖真值分支。在表 2-10 中给出的测试用例虽然能做到分支覆盖，但却不能做到条件覆盖。

表 2-10　没有达到条件覆盖的分支覆盖测试用例集

测试用例	A	B	C
1	False	False	False
2	True	False	False

条件覆盖只是让每个条件的真假值在测试用例中出现一次，而不同条件真假值之间相互作用无法检测，因此有必要设计测试用例来覆盖各个条件之间各种可能的取值组合，称之为条件组合覆盖。

习题 2.1.6

分支条件覆盖与分支覆盖和条件覆盖是否具有强弱关系？

2.1.7　条件组合覆盖测试（Branch Condition Combination Testing）

设计测试用例让判定语句中的每个条件变量的真假值组合情况都可执行到。表 2-11 列出了满足图 2-7 中程序例子的条件组合测试用例。判定语句中涉及 3 个布尔变量，组合起来有 8 条测试用例。条件组合测试是最彻底的测试，n 个条件变量就需要 2^n 个测试用

例，这在变量多的情况下，测试代价将会非常高。

表 2-11　条件组合覆盖测试用例集

测试用例	A	B	C
1	False	False	False
2	True	False	False
3	False	True	False
4	False	False	True
5	True	True	False
6	False	True	True
7	True	False	True
8	True	True	True

条件组合覆盖一定强于分支条件覆盖，因为条件组合覆盖已经覆盖了各个条件之间的所有可能组合，自然也就包含了这些条件形成的所有可能的判定所对应的真假值分支。作为最强的测试覆盖，条件组合测试需要的测试用例随着条件个数 n 的增长呈指数速度 2^n 增长，当 n 很大时，测试成本几乎无法承受。因此，在下一节介绍一种几乎等效的，但成本却非常小的测试覆盖准则。

习题 2.1.7

条件组合测试的优缺点分别是什么？

2.1.8　修改决策条件测试（Modified Condition Decision Coverage Testing）

设计测试用例，让每个条件变量独立改变判定语句的真假值。这种测试方法的优点是：保证每个变量真假值出现一次；整个判定表达式出现真假值；让每个变量独立影响判定表达式的真假值；测试用例最好的情况下规模为 $n+1$，最差情况下规模为 2^n。相对于条件组合覆盖需要 2^n 条测试用例，修改决策条件测试（MC/DC）是最实用的一种测试方法。

例如，针对图 2-7 中的程序例子，表 2-12 给出了两条测试用例 A1 和 A2，它们让条件 A 独立改变判定表达式的值；表 2-13 给出了两条测试用例 B1 和 B2，它们让条件 B 独立改变判定表达式的值；表 2-14 给出了两条测试用例 C1 和 C2，它们让条件 C 独立改变判定表达式的值。表 2-12、表 2-13 和表 2-14 中的测试用例合起来就是表 2-15，因为 A1 和 B1 相同，B2 和 C1 相同。

表 2-12　条件 A 独立改变判定表达式的值

测试用例	A	B	C	输出
A1	False	False	True	False
A2	True	False	True	True

表 2-13 条件 B 独立改变判定表达式的值

测试用例	A	B	C	输出
B1	False	False	True	False
B2	False	True	True	True

表 2-14 条件 C 独立改变判定表达式的值

测试用例	A	B	C	输出
C1	False	True	True	True
C2	False	True	False	False

表 2-15 修改决策条件测试用例集

测试用例	A	B	C	输出
1(A1,B1)	False	False	True	False
2(A2)	True	False	True	True
3(B2,C1)	False	True	True	True
4(C2)	False	True	False	False

习题 2.1.8

修改决策条件测试具有哪些优点?

2.1.9 路径覆盖测试(Path Testing)

路径测试就是设计足够的测试用例,覆盖程序中所有可能的路径。以图 2-8 为例,则可以选择如表 2-16 所示的一组测试用例来覆盖该程序段的全部路径。

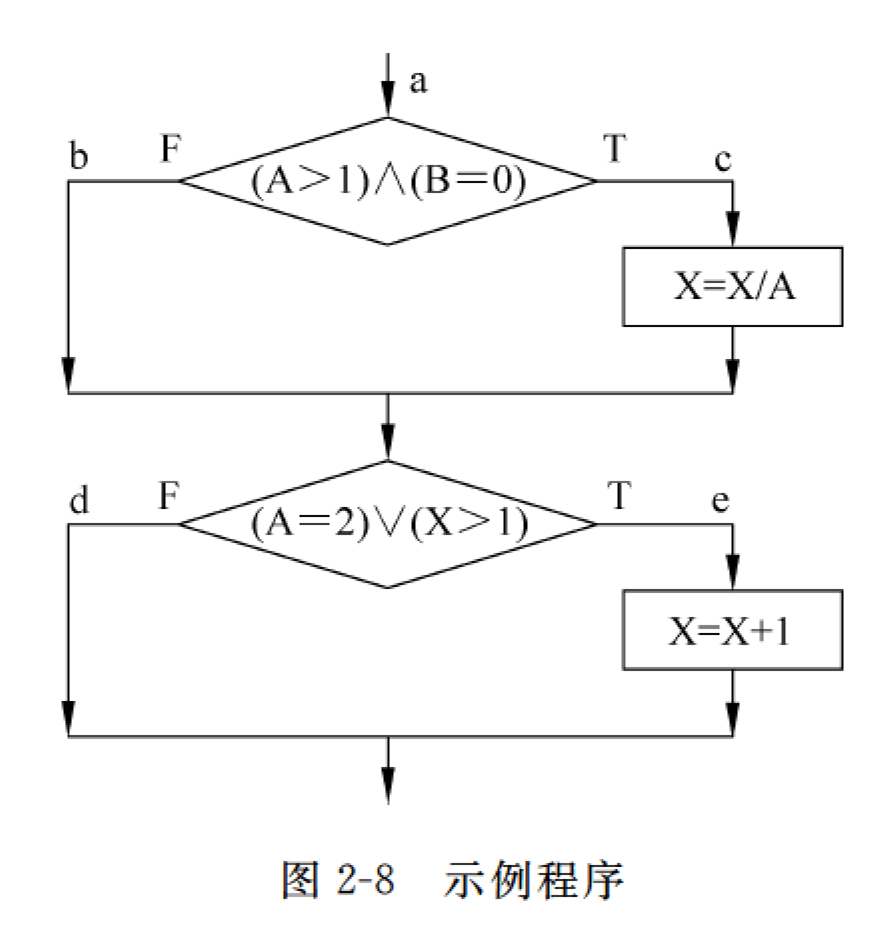


图 2-8 示例程序

路径覆盖测试方法的成本非常高,因为在一些程序中路径的总数非常大,甚至是无穷的。有一种路径测试的简化方法叫基本路径测试,该方法是在程序控制流图的基础上,通过分析控制构造的环境复杂性,导出一组最少的基本可执行路径集合,这些基本路径之间不能互相包含,任何一条路径要包含一个独立的部分,这个独立部分在其他路径中都不包含。然后根据这些基本路径来设计测试用例,设计出的测试用例要保证在测试中程序的每一个可执行语句至少执行一次。如图 2-8 所示的路径 ace 和 abd 就是该示例程序的两条基本路径。

表 2-16 路径覆盖测试用例集

测 试 用 例	通 过 路 径	覆 盖 条 件
【(2,0,4),(2,0,3)】	ace	A>1 B=0 A=2 X>1
【(2,1,1),(2,1,2)】	abd	A>1 B≠0 A≠2 X≤1
【(1,0,3),(1,0,4)】	abe	A>1 B≠0 A=2 X>1
【(3,0,3),(3,0,1)】	acd	A>1 B=0 A≠2 X≤1

习题 2.1.9

什么叫基本路径覆盖？它与路径覆盖的区别是什么？

2.1.10 线性代码序列跳转测试(LCSAJ Testing)

线性代码序列跳转测试主要关注程序控制流的跳转。一个线性代码序列跳转表示为一个三元组：线性代码起点行数 s、线性代码终点行数 e 和控制流跳转点的代码行数 j，即(s,e,j)。测试用例用来执行覆盖每个线性代码序列跳转，每个测试用例可以描述为三个部分：测试输入、被该测试用例执行的线性代码序列跳转、预期输出。由于线性代码序列跳转测试的对象是线性代码序列跳转，因此线性代码序列跳转测试的覆盖率度量为

$$\text{LCSAJ} = \frac{\text{执行过的线性代码序列跳转个数}}{\text{所有的线性代码序列跳转个数}} \times 100\%$$

以下通过判断一个整数是否是素数的 BASIC 程序例子(如图 2-9 所示)来说明线性代码序列跳转测试的有关概念和方法。图 2-10 列出了图 2-9 中程序所包含的线性代码序列跳转及相应的条件。

```
READ (Num);
WHILE NOT End of File DO
        Prime := TRUE;
        FOR Factor := 2 TO Num DIV 2 DO
                IF Num - (Num DIV Factor)*Factor = 0 THEN
                        WRITE (Factor, ' is a factor of', Num);
                        Prime := FALSE;
                ENDIF;
        ENDFOR;
        IF Prime = TRUE THEN
                WRITE (Num, ' is prime');
        ENDIF;
        READ (Num);
ENDWHILE;
WRITE ('End of prime number program');
```

图 2-9 素数判断程序

例 2-2 判断一个整数是否是素数。

首先设计一个测试用例如表 2-17 所示，该测试用例输入 3 个数：5、6 和 2。执行这个测试用例之后，表 2-18 列出了各个线性代码序列跳转的执行情况。可以发现很多线性代码序列跳转还没有被执行，需要补充一些测试用例。

(2->3) : requires NOT End of File
(2->15) : Jump requires End of File
(4->5) : requires the loop to execute, i.e. Num DIV 2 is greater than or equal to 2
(4->10) : Jump requires the loop to be a zero-trip, i.e. Num DIV 2 is less than 2
(5->6) : requires the IF statement on line 5 to be true
(5->9) : Jump requires the IF statement on line 5 to be false
(9->5) : Jump requires a further iteration of the FOR loop to take place
(9->10) : requires the FOR loop to have exhausted
(10->11) : requires Prime to be true
(10->13) : Jump requires Prime to be false
(14->2) : Jump must always take place

图 2-10 程序中包含的线性代码序列跳转及相应的条件

表 2-17 第一个测试用例

测试用例	输 入	期望输出
1	5	5 is prime
	6	2 is a factor of 6
		3 is a factor of 6
	2	2 is prime
		End of prime number program

表 2-18 各个线性代码序列跳转的执行情况

LCSAJ			
起 始 行	终 止 行	跳转到行	执行次数
1	2	15	0 ***
1	4	10	0 ***
1	5	9	1
1	9	5	0 ***
1	10	13	0 ***
1	14	2	0 ***
2	2	15	1
2	4	10	1
2	5	9	0 ***
2	9	5	1
2	10	13	0 ***
2	14	2	0 ***
5	5	9	0 ***
5	9	5	0 ***
5	10	13	1
5	14	2	0 ***
9	9	5	0 ***
9	10	13	0 ***
9	14	2	1
10	10	13	0 ***

续表

起　始　行	终　止　行	跳 转 到 行	执 行 次 数
10	14	2	1
13	14	2	1
15	15	exit	1
LCSAJ 的数量			23
已执行数量			9
未执行的数量			14
LCSAJ 的覆盖率			39%

图 2-11 给出了各个未被执行的线性代码序列跳转及其需要满足的执行条件，根据这个条件，可以设计如表 2-19 所示的测试用例。另外，执行了代码行 7 就无法执行代码行 11，因此线性代码序列跳转(1,14,2)、(2,14,2)、(5,14,2)是不可能的。(10,10,13)也是不可能的线性代码序列跳转，因为这个跳转需要语句行 10 的执行结果是 prime＝false，而该线性代码序列跳转的起点是语句 10，之前必须是从语句 4 跳转而来(如果从语句 9 过来，则矛盾)，从语句 4 跳转过来，意味着语句 3 一定已经被执行，即 prime＝true 已被赋值，所以此时执行语句 10，一定不会从语句 10 跳转到语句 13。

LCSAJ	Comments
(1, 2, 15)	A new test is needed with no data, so the End of File is found immediately.
(1, 4, 10)	A new test is needed with a number less than 4 as first in the list.
(1, 9, 5)	A new test is needed with an even number greater than 5 as first in the list.
(1, 10, 13)	A new test is needed with the number 4 as first in the list.
(2, 5, 9)	This will be executed with an odd number greater than 4 which is not the first in the list.
(2, 10, 13)	This will be executed with the number 4 which is not the first in the list.
(5, 5, 9)	This will be executed by a number greater than 6.
(5, 9, 5)	This will be executed by an odd non-prime number greater than 7.
(9, 9, 5)	This will be executed by a number greater than 7.
(9, 10, 13)	This will be executed by an odd non-prime number greater than 8.

图 2-11　各个未被执行的线性代码序列跳转及其需要满足的执行条件

表 2-19　新设计的测试用例

测试用例	输　　入	预期输出	已执行 LCSAJS
2	＜none＞	End of prime number program	(1,2,15)
3	2	2 is prime	(1,4,10)
	4	2 is a factor of 4	(2,10,13)
		End of prime number program	
4	8	2 is a factor of 8	(1,9,5)
		4 is a factor of 8	(5,5,9)
		End of prime number program	
5	4	2 is a factor of 4	(1,10,13)
	11	11 is prime	(2,5,9)
			(9,9,5)
			(5,5,9)
			(9,10,13)
		End of prime number program	

习题 2.1.10

线性代码序列跳转测试与分支覆盖的区别是什么?

2.1.11 小结

前面介绍了 10 种白盒动态测试方法,这 10 种方法之间既有着密切的关系,又有很大的区别。路径测试(All Paths)和分支条件组合方法(BCC)是两种最强的测试方法,但它们之间不存在相互包含关系。图 2-12 中箭头连接的两种方法之间存在相互包含关系,例如,BCC 包含 MCDC,即 BCC 测试之后,一定满足 MCDC 的测试要求,反之,不一定。

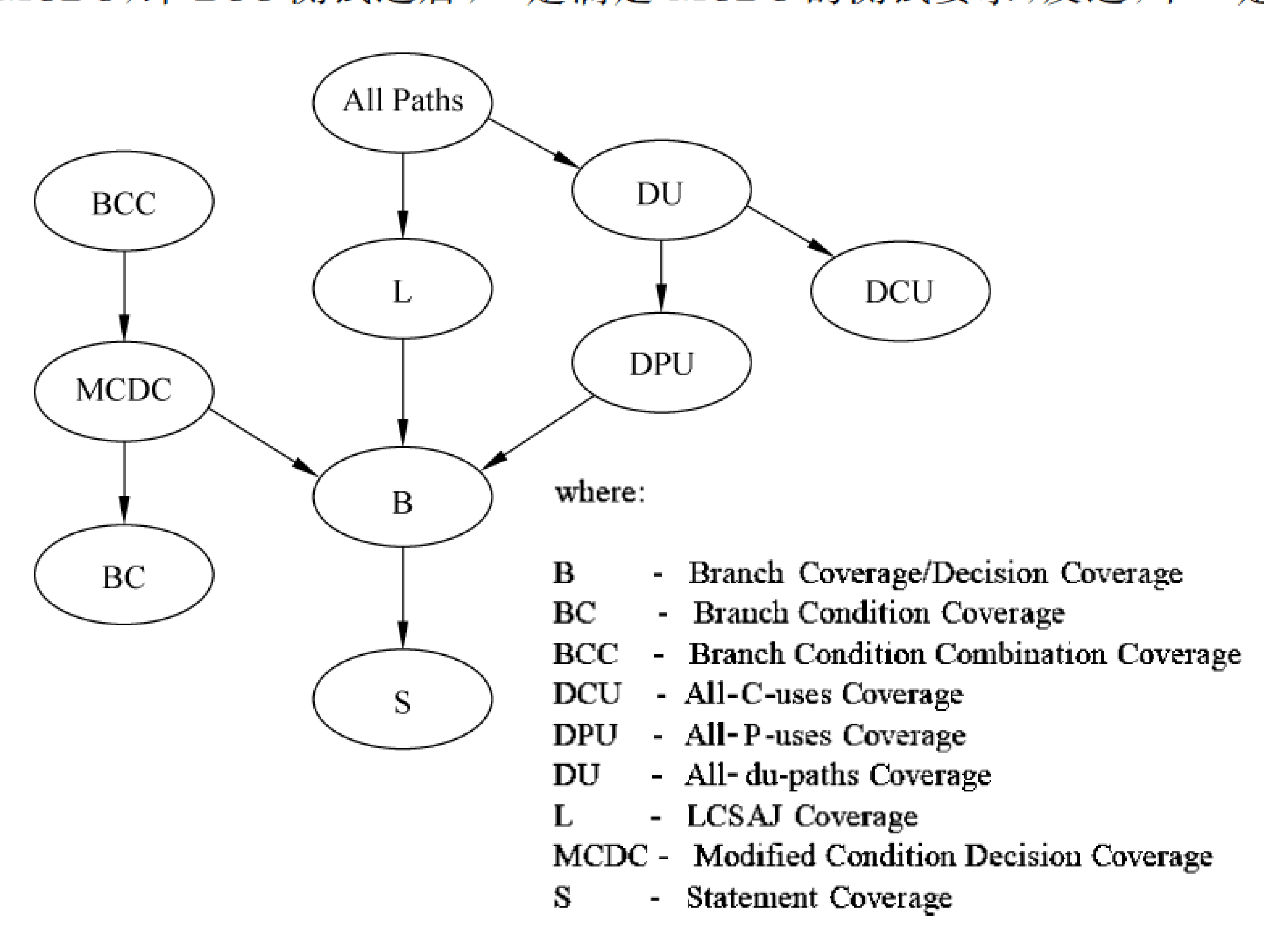


图 2-12 各种白盒测试方法相互关系

习题 2.1.11

1. 用 C 语言设计一个三角形分类程序,输入 3 个正数表示三角形三条边的长度,判断该三角形是等边三角形、等腰三角形还是普通三角形或不是三角形。设计测试用例覆盖该程序的每个分支。

2. 用 C 语言编写一个可以输出两个整数的最大公因数的程序,例如,输入两个正整数 16 和 36,程序可以给出它们的最大公因数是 4。设计测试用例覆盖该程序中所有语句。

2.2 黑盒测试(Black Box Testing)

1. 概念

所谓黑盒测试是指把待测试程序或软件系统看成一个无法打开的“黑匣子”(如图 2-13 所示),在完全不知道程序的内部结构和处理过程的前提下,在程序接口进行的测试。它只

检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接受输入数据并产生正确的输出信息，并且保持外部信息的完整性。因此，黑盒测试又称为功能测试或数据驱动的测试。

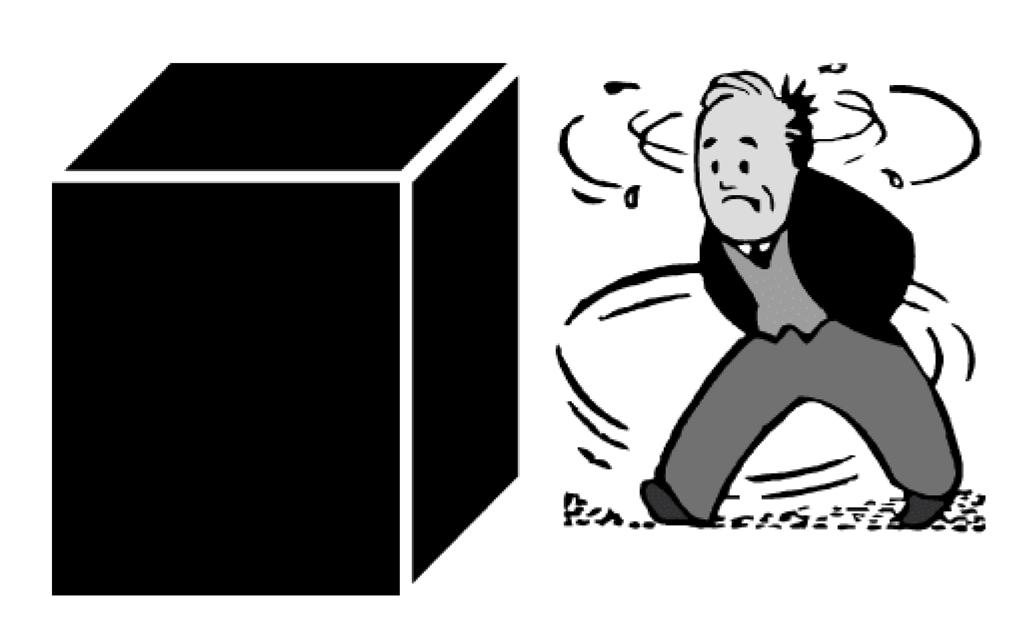


图 2-13　黑盒测试示意图

黑盒测试还包括非功能性测试，例如性能测试，检查软件各项功能的性能是否达到相应的指标；强度测试，检查软件在各种极端条件下是否可以正常工作。还有安全性测试、安装与卸载测试、配置测试、兼容性测试、故障修复测试、可使用性测试和帮助测试等都可以归为非功能性的黑盒测试。

黑盒测试也分静态和动态。静态黑盒测试是指对软件规格说明书的审查，它不执行软件。动态黑盒测试是运行待测试软件，检查其功能的正确性和完整性等。所有的测试可以根据是否执行程序，是否理解内部实现分成如下类型（如表 2-20 所示）。

表 2-20　测试按照是否执行及是否依据内部结构进行的分类

	黑盒测试：基于规格说明的测试	**灰盒测试：结合代码实现和规格说明的测试**	**白盒测试：基于程序代码和实现的测试**
静态测试：不执行程序	静态-黑盒：规格说明文档审查	静态-灰盒：同时审查规格说明和相应的代码实现	静态-白盒：代码审查、走查等
动态测试：执行程序	动态-黑盒：根据规格说明设计并运行测试	动态-灰盒：结合代码实现和规格说明设计并运行测试	动态-白盒：运行程序，实现结构覆盖

2．目标

黑盒测试主要为发现以下几类错误：

（1）检查程序功能是否按照需求规格说明书正常使用？是否有不正确和遗漏的功能？

（2）检查人机交互是否错误？输入是否被正确接受？是否输出正确的结果？

（3）是否有数据结构错误或外部信息访问错误？

（4）检测软件运行性能上是否满足要求？

（5）是否存在初始化或终止性错误？

3. 原理

对软件进行黑盒测试，测试者并不知道被测试对象的内部是如何构造以及如何工作的，只基于所知的系统需求来实施测试。根据黑盒测试的概念，我们可知，黑盒测试并不只是一种测试技术，而是一类不需要知道内部设计或代码信息的测试技术总称。

黑盒测试不是用来替代白盒测试等其他测试方法，而是用来作为这些测试方法的补充。黑盒测试不仅是功能测试，还包含非功能性测试，注重测试软件的功能性及非功能性需求，需要根据软件规格说明书，利用等价类划分方法、边界值分析方法、因果图分析方法、状态转换图和错误猜测等方法为测试程序生成各种测试输入，检测程序所有功能及非功能性需求。

黑盒测试一般由有经验的软件测试人员在验收测试和系统测试等软件测试阶段进行。检查软件系统的功能是否被正确实现，如界面是否正确、外部数据库访问是否有问题、性能是否符合要求、系统初始化和终止是否有问题等；检查系统实现是否有功能遗漏等。

4. 方法

通过认真阅读和检查软件规格说明书，一方面发现规格说明书中可能存在的问题，另一方面在充分理解规格说明书的基础上，对软件输入进行等价类划分和边界值分析，构建因果图、状态转换图以及输入语法等，并对系统进行适当的错误猜测，从而进行如下测试(黑盒测试方法流程如图 2-14 所示)。

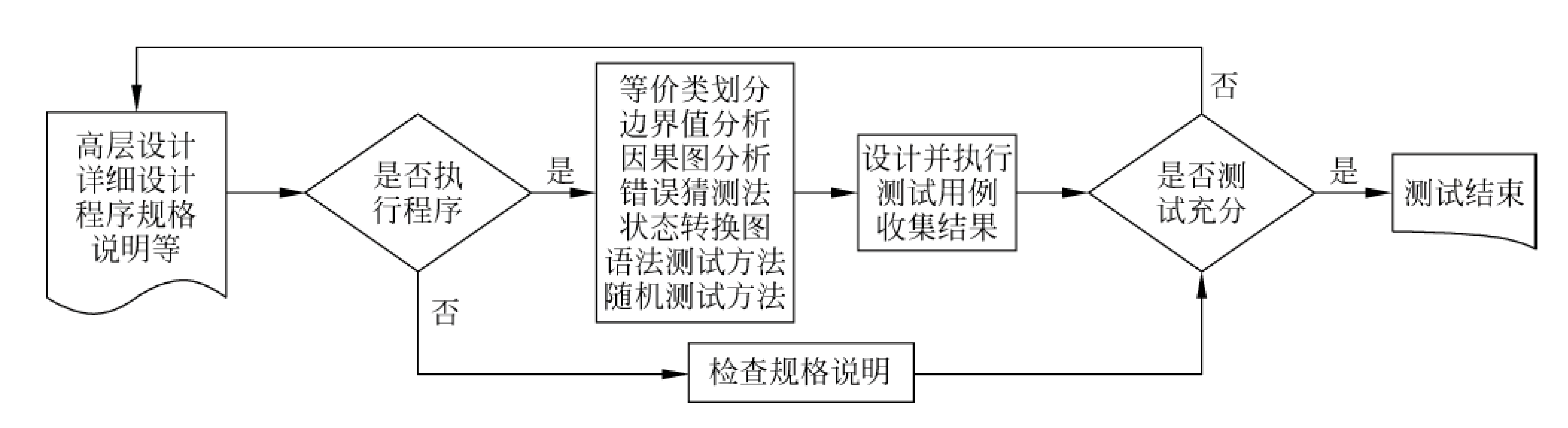


图 2-14 黑盒测试方法的流程图

对软件进行黑盒测试，一定要结合被测试软件的特点，选择性地使用以上方法，并将它们有机地结合起来。例如，按照待测试软件的输入输出条件进行等价类划分，通过在等价类中选择测试用例，可以首先将无限的测试域变为有限，同时等价类划分也是边界值分析方法的基础。应用边界值分析可以找出一些容易发现错误的测试输入，这一点在人们在实践中经常得到验证。应用错误猜测方法，结合测试人员的直觉和经验选择一些测试用例可以作为重要的补充。如果软件的功能规格说明中含有输入条件的组合情况，就应使用因果图分析方法；如果其中有非常明显的状态转换特征，就应采用状态转换测试方法(如表 2-21 所示)。

表 2-21　各种黑盒测试方法的对比

	等价类划分	边界值分析	因果图分析	错误猜测法	状态转换图	语法测试方法	随机测试方法
适用场景	测试空间无限大	测试输入有明确范围	软件条件间存在关系	任何场景补充方法	软件状态特征明显	在对输入进行确认时发现不足之处	任何场景补充方法
适用目的	选择有限测试用例充分测试，避免冗余	发现边界上的问题	发现输入条件之间关系上的问题	建立可能的错误检查表	发现状态转换中可能出现的问题	发现不识别、不正确的输入	发现意料之外的错误，无任何技术要求
使用原理	在等效的测试用例集合中选择代表	实践反复证明：边界值容易发现问题	将需求规格说明书转化为决策表的系统化方法	测试人员的经验和直觉是有价值的	采用状态转换图来描述软件运行，进行图测试	基于输入接口语法变异产生测试用例	随机生成测试输入

很多种测试方法都可划分为黑盒测试，例如，组合测试、统计测试、基于模型的测试和基于性质的软件测试方法。

5. 实例

黑盒测试面临的问题与白盒测试一样，那就是它有一个庞大的可用测试用例。例如，在三角形分类程序中，需要 3 个整数型的输入数据，如果计算机字长为 16 位，则每个整数可能取的值有 2^{16} 个，3 个输入数据的各种可能值的排列组合共有：

$$2^{16} \times 2^{16} \times 2^{16} = 2^{48} \approx 3 \times 10^{14}(\text{种})$$

也就是说，大约需要把这个程序执行 3×10^{14} 次才能做到“穷尽”测试。假定每执行一次程序需要 1ms，执行这么多次大约需要 1 万年；不仅测试时间长得叫人不可思议，测试得出的输出数据更是多得完全无法分析。黑盒测试就是研究如何在庞大的可用测试空间中选择少量的测试用例，进行科学而有效的测试。

6. 优缺点

现在软件越来越复杂，功能越来越强，了解软件实现细节越来越难。黑盒测试不需要具体的编程语言细节，可以将测试工程师的精力主要集中在根据功能需求来设计测试用例，验证软件是否按照预期要求工作。黑盒测试不要求软件测试人员掌握关于软件实现的任何知识，甚至可以不会编程语言。

由于黑盒测试技术不需要源代码，在源代码无法得到或者源代码很复杂、难以阅读的情况下，黑盒测试技术仍可以进行有效的测试。黑盒测试针对最终用户的观点来执行，一般是公平的，因为测试人员和开发人员相互独立。

功能性与软件如何实现无关，因此如果实现发生变化，黑盒测试用例仍然有用。而且黑盒测试用例与软件开发实现可以并行进行，一旦功能性规格说明完成，测试人员就可以设计测试用例，而开发人员则可以根据规格说明开始编程实现，可以有效压缩总的项目开发时间。黑盒测试有助于识别功能性规格说明中含糊、矛盾的部分。

黑盒测试不能保证代码和路径等结构覆盖率，可能有些代码和路径等结构无法得到测试，不能直接对可能隐藏了许多问题的特定程序段进行测试；在没有清晰、准确的规格说明时，难以设计恰当的测试用例，甚至有些错误无法发现，形成一些软件漏洞。黑盒测试的测

试用例之间往往存在一些冗余。

习题 2.2

1. 黑盒测试有哪些具体的方法，这些方法之间有什么互补关系？
2. 每种黑盒测试方法都有各自的特点，如何有效地综合使用各种黑盒测试方法？
3. 黑盒测试方法有哪些优点和不足？

2.2.1 等价类划分（Equivalence Class Partition）

1. 概念

等价类是指某个输入域的集合，在这个集合中每个输入条件都是等效的。等价类划分法认为：如果使用等价类中一个条件作为测试数据进行测试不能发现程序的缺陷，那么使用等价类中的其他条件进行测试也不会发现错误。等价类划分法是一种典型的黑盒测试方法，它完全不考虑程序的内部结构，只根据需求规格说明书对输入范围进行划分，把所有可能的输入数据，即程序输入域划分为若干个互不相交的子集，称为等价类，然后从等价类中选取少数具有代表性的数据作为测试用例进行测试。

2. 目标

待测试软件的输入空间太大，穷举测试无法进行，只能在巨大的可能数据空间中选择很少一部分进行测试。使用等价类划分方法作为功能性测试的测试用例选择方法，目标有两个：希望进行完备的测试，同时又希望避免冗余。

3. 方法

在使用等价类划分方法进行测试时，首先应在分析需求规格说明的基础上划分等价类，再根据等价类设计出测试用例。如何确定等价类，是等价类划分方法的一个重要问题。以下给出几条确定等价类的原则：

（1）按区间划分。如果规格说明规定了输入条件的取值范围或值的数量，则可以确定一个有效等价类和两个无效等价类。例如，如果软件规格说明“学生允许选修 5 到 8 门课”，则一个有效等价类可取“选课 5 到 8 门”，无效等价类可取“选课不足 5 门”和“选课超过 8 门”。

（2）如果输入条件规定了输入数据的一组可能的值，而且程序是以不同的方式来处理每一种值，则可将每一种值划分为一个有效等价类，并划分一个无效等价类。例如，程序输入 x 取值为枚举类型{2,4,6,8}，且程序中对这 4 个数值分别进行了处理，则有效等价类为 x=2，x=4，x=6，x=8，无效等价类为 x≠2，4，6，8。

（3）按照数值集合划分。在输入条件规定了输入集合或规定了“必须如何”的条件下，可以确定一个有效等价类和一个无效等价类（该集合有效字值之外）。例如，程序输入条件为正的奇数，则有效等价类为正奇数，无效等价类为其他数。

（4）按照限制条件或规则划分。在规定了输入数据必须遵守的规则或限制条件下，可确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。例如，程序

输入条件为以字符 a 开头、长度为 8 的字符串，并且字符串不包含 a～z 之外的其他字符，则有效等价类为满足了上述所有条件的字符串，无效等价类为不以 a 开头的字符串、长度不为 8 的字符串和包含了 a～z 之外的其他字符的串。

等价类划分过程中可用表 2-22 形式形成一个记录清单。

表 2-22 记录清单

输入条件	有效等价类	无效等价类
……	……	……

根据等价类表，就可以确定测试用例。首先，为每个等价类确定一个唯一的编号，然后，设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类；重复这一步，直到所有有效等价类均被测试用例覆盖；最后，设计测试用例逐一覆盖每个无效等价类。值得注意的是，无效等价类需要逐一设计测试用例进行覆盖，因为几个无效等价类在一起进行测试会相互屏蔽。

等价类划分测试技术的测试准则（等价类划分覆盖率）可以定义为执行等价类的数量与总共描述的等价类数量之比：

等价类划分覆盖率 =（执行的等价类数量/总的等价类数量）× 100%

4. 原理

所谓等价类是指输入域的一组互不相交的子集，该组子集的并是整个输入域，因此依据等价类划分方法设计的测试用例具有完备性和无冗余性。由于等价类由等价关系决定，因此等价类中的元素有一些共同特点：如果用等价类中的一个元素作为测试数据不能发现程序中的故障，那么使用集合中的其他元素进行测试也不能发现故障。对于揭露软件错误来说，等价类中的每个元素应该是等效的，如果测试数据都从一个等价类中选取，会造成浪费，因为只有其中一个测试数据对发现错误有效，所以有必要将测试用例分布到各个等价类中。

软件不能只接受有效、合理的数据，还要经受意外的考验，即接收无效的或不合理的数据，这样获得的软件才能具有较高的可靠性。因此，在考虑等价类时，应注意区别两种不同的情况：

（1）有效等价类。有效等价类是指符合需求规格说明书，有意义的、合理的输入数据所构成的集合。利用有效等价类，可以检验程序是否实现了规格说明预先规定的功能和性能。在具体问题中，有效等价类可以是一个，也可以是多个。

（2）无效等价类。无效等价类是指不符合需求规格说明书，不合理或无意义的输入数据所构成的集合。利用无效等价类，可以检查软件功能和性能的实现是否有不符合规格说明的地方。对于具体的问题，无效等价类至少应有一个，也可能有多个。

5. 实例

例 2-3 三角形问题。

输入 3 个整数 a、b、c 作为边，程序的输出是由这 3 条边确定的三角形类型：等边三角

形、等腰三角形、不等边三角形或非三角形，如表 2-23 所示。

表 2-23 例 2-3 用表

输入条件	有效等价类	无效等价类
a、b、c	a>0、b>0、c>0	a≤0、b≤0、c≤0
a、b、c	a+b>c、b+c>a、a+c>b	a+b≤c、b+c≤a、a+c≤b

例 2-4 NextData 函数。

NextData 函数有 3 个输入变量(月份 m、日期 d 和年 y)，返回输入日期后面的那个日期，如表 2-24 所示。

表 2-24 例 2-4 用表

输入条件	有效等价类	无效等价类
m	1≤m≤12	m<1、m>12
d	1≤d≤31	d<1、d>31
y	1900≤y≤2100	y<1900、y>2100

6. 优缺点

等价类划分方法是一种科学有效的方法，该方法将无限的输入空间约简为小规模的有限集合，理论上这两个集合对于待测试软件是等效的。在有明确的条件和限制的情况下，利用等价类划分技术可以设计出完备的测试，减少不必要的测试用例。

等价类划分法的优点是考虑了单个输入域的各类情况，避免了盲目或随机选取输入数据的不完整性和覆盖的不稳定性。尽管等价划分法要比随机选取测试用例优越得多，但它仍然存在不足。例如，这种方法可能只考虑原子(部分)条件的输入输出，那么条件之间的相关性或相互作用就有可能被忽略；如果它们之间的关系考虑在内，则代价又会非常高。对于这个问题，可以先对各个原子条件进行等价类划分，再对其进行组合的方法做到这一点。等价类方法还可能忽略掉了某些特定类型的高效测试用例，边界值分析和因果图可以弥补相应的不足。

习题 2.2.1

1. 等价类划分的意义是什么？
2. 给出以下佣金问题的等价类划分。

佣金问题：

一位步枪销售商销售每个枪机 45 美元，每个枪托 30 美元，每个枪管 25 美元，销售商每月至少要售出一只完整的步枪，且生产限额是大多数销售商在一个月内可销售 70 个枪机、80 个枪托和 90 个枪管。销售商每访问一个镇子之后，都给生产商发电报，说明在那个镇子售出的枪机、枪托和枪管的数量。到了月末，销售商要发出“枪机＝－1”表示一个月结束。这样制造商就知道当月的销售情况，并计算销售商的佣金如下：销售不到(含)1000 美元的部分为 10%，1000(不含)～1800(含)美元的部分为 15%，超过 1800 美元的部分为 20%。佣金程序生成月份销售报告，汇总售出的枪机、枪托和枪管总数，销售商的总销售额以及佣金。

2.2.2 边界值分析(Boundary Value Analysis)

1. 概念

软件测试实践中，大量的错误往往发生在输入或输出范围的边界上，而不是在内部。例如，数组的下标、循环控制变量等边界附近往往出现大量错误。因此，作为等价类划分方法的补充，边界值分析方法不是选择等价类的任意元素，而主要针对各种边界情况设计测试用例。

2. 目标

大量测试实践表明，很多错误发生在输入或输出数据范围的边界上，因此，针对各种边界情况设计测试用例，有利于揭露程序中的错误。

3. 方法

边界值分析关注的是输入输出空间的边界条件，用以选择测试用例。实践表明，程序在处理大量中间数值时都正确，但在边界处可能出现错误。例如，循环条件漏写了等于，计数器少计了一次或多计了一次，数组下标忽略了 0 的处理等，这些都是平时编程容易疏忽而导致出错的地方。

一些可能与边界有关的数据类型有数值、速度、字符、位置、尺寸、数量等，针对这些数据类型可以考虑它们的下述特征：第一个和最后一个、最小和最大、开始和完成、超过和在内、空和满、最短和最长、最慢和最快、最高和最低、相邻和最远等。在具体应用中，每个软件都可能包含各式各样的边界条件，应视具体情况而定。

可以预先定义期望的边界值覆盖率，并在执行测试后对覆盖率进行计算：

$$边界值覆盖率 = (执行的边界值数量/总的边界值数量) \times 100\%$$

其中，边界值数量必须考虑边界值上限的临近数值和下限的临近数值的数量。当然，具体的数量只包含不相等的输入值，对于临近等价类的重叠数据只作为一个边界值来看待，因为对于每个测试数据都只有一个测试用例与之对应。

4. 原理

边界值分析是对从等价类划分导出的测试用例的一个非常合理的补充。在程序中错误经常出现在等价类的边界上，出现这种现象的原因是经常没有明确定义边界值，或者编程人员对边界值产生误解。利用边界值进行的测试往往能够有效地发现失效，边界值分析方法一般要求各等价类中的数据有明确边界值。

测试软件的边界值，就好像让人行走在悬崖边上，如果一个人能在陡峭、险峻的悬崖边安全地正常行走，那么他就能在几乎任何陆地上正常行走。如果软件在各种边界值上正常运行，那么它就能正常运行在各种情况下。

5. 实例

这里先看一个小程序。

```
int i, data[10];
For (i=1;i<=10;i++) Data[i]=-1;
```

该小程序中有个 Bug，申请的整数数组是从 data[0]开始的，程序中 data[0]未被初始化，而data[10]不在已申请的数组范围之列。

对于一个输入域，必须考虑边界值和处于输入域之外的临近值。比如，输入域为[−1.0，1.0]，测试数据应为：−1.0，1.0，−1.001，1.001。

若输入文件的数据记录有限，可以根据以下方法考虑边界值：假如一个文件的记录个数在 1～100，则测试数据应该是 1 和 100 以及 0 和 101。

假如以输出域作为测试基础，可以按照下面的规则进行分析：测试对象的输出是整型值，范围是 500～1000，得到的测试输出应该是 500、1000、499 和 1001。这里需要花费一定的工作量来标识得到相应输出的每个输入值。产生无效的输出可能不现实，但尽量试试也许能发现一些缺陷。

假如测试的是允许的输出值的数量，则处理的方式和测试允许的输入值的数量的方式一样：假如输出允许的是 1～4 个数据，测试需要产生的输出值是 1 和 4 以及 0 和 5。

对于有序集合，测试特别感兴趣的元素是第一个和最后一个。

假如以复杂的数据结构作为输入或输出，那么一个空的列表和 0 矩阵等可以作为边界值。

对于数值计算，靠得近的数据和距离比较远的数据可以作为边界值。

对于无效等价类，只有当无效等价类内的值可以触发测试对象一个不同的异常处理时，进行边界值的分析才有意义。

应尽量选择非常庞大的数据结构、列表和表格等作为边界值分析的数据，比如，那些能使内存溢出、文件和数据存储到达边界的数据，来检查测试对象在这种极端情形下的行为。

对于列表和表格，空列表和满列表以及列表的第一个元素和最后一个元素都是应该作为分析的对象，因为测试它们常常可以发现由于编程错误而导致的失效。

6. 优缺点

边界值分析应该与等价类划分一起使用，因为在等价类边界值上发现问题的概率比在等价类内部发现问题的概率高得多。这两种技术很容易相互结合着使用，同时在选择具体测试数据上可以保持足够的独立性。

利用边界值分析技术来定义相应的测试用例需要很多的创造性，这一点经常会被忽略，因为这种技术看起来似乎很简单，而实际上确定相关的边界值需要一些经验和专门的知识。

习题 2.2.2

为什么测试用例设计中，边界值分析很重要？

2.2.3 因果图和决策表（Cause Effect Graph and Decision Table）

1. 概念

因果图是一种形式语言。用自然语言描述的规格说明可以转换为因果图，因果图相当于一种数字电路，一个组合的逻辑网络，但没有使用标准的电子学符号，而是使用了简化的

符号。因果图是一种可以辅助测试者明确测试对象，确定测试依据的有力手段。该方法很好地弥补了边界值分析和等价类划分中未对输入条件的组合进行分析的弱点。

因果图必须转化成决策表，从决策表才可以得到相应的测试用例。决策表由两部分组成，上半部分罗列了输入（原因），下半部分包含了结果。每一列都是一个测试用例，条件的组合以及针对这一组合的期望结果或输出。

因果图和决策表技术是一种黑盒测试方法，从分析软件系统需求规格说明书开始，得到因果列表，然后基于此列表建立决策表，最后基于决策表生成测试用例。

2. 目标

在等价类和边界值分析方法中，对输入条件的考虑并未重视输入条件的组合。事实上，当输入条件存在若干可能的组合时，必须对这些组合加以考虑，以证实测试程序在某种输入组合的情况下能否完成规格说明书中预先规定的功能。

要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，它们之间的组合情况也相当多。因此必须考虑使用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计测试用例，这就需要利用因果图。因果图方法最终生成的就是决策表，它适于检查程序输入条件的各种组合情况。基于决策表测试设计的测试用例的目的是执行感兴趣的输入组合。令人感兴趣的点主要是指可能发现问题的点。除了原因和结果之外，决策表还可能包含中间结果。

3. 原理

因果图方法是一种根据条件的组合而生成测试用例的系统性的方法。可以替代这种方法的是特殊选取的条件组合，但在这个过程中，很可能会遗漏很多可由因果图方法确定的“令人感兴趣”的测试用例。

由于因果图方法需要将规格说明转换为一个布尔逻辑网络，因此它使我们从不同的视角，更细致、深入地来审视规格说明。事实上，建立因果图是一个暴露规格说明中模糊和不完整之处的好方法。

从因果图产生的决策表的每一列都可以清楚地看到条件（原因）及其输入的依赖关系，以及由这些输入组合得到的相应的输出和结果。决策表定义了逻辑测试用例，为了执行这些测试用例，必须输入具体的数据值并且标识前置条件和后置条件。

可以定义基于因果图产生的决策表的测试完成准则。最基本的要求是至少用一个测试用例来执行决策表中的每一列，这样就验证了所有关心的输入条件组合和相应的输出结果。

不太乐观的是，原因的每个组合在设计成一个测试用例时，条件可能会相互影响，或者相互排斥，因此不是所有的组合都是有效的。

4. 方法

1）用因果图法生成测试用例的基本步骤

（1）分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件）。所谓原因，是指输入条件或输入条件的等价类，而结果是指输出条件。给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明描述中的语义,找出原因和结果之间、原因与原因之间对应的关系。根据这些关系,画出因果图。

(3) 由于语法或环境限制,有些原因和原因之间、原因和结果之间的组合情况不可能出现。为表明这些特殊的情况,在因果图上用一些记号标明约束或限制条件。

(4) 把因果图转换成决策表。因果图必须转化成决策表,从决策表才可以得到相应的测试用例,将因果图转化为决策表的步骤如下。

① 选择结果。

② 根据因果图查找能够得到这个结果的原因组合,以及不产生这个结果的原因组合。

③ 在决策表中为每一个原因组合以及引起这个结果的状态加一列。

④ 检查决策表条目是否出现冗余,如果是,则删除那些冗余的条目。

(5) 把判定表的每一列拿出来作为依据,设计测试用例。

2) 在因果图中出现的基本符号

通常在因果图中用 C_i 表示原因,用 E_i 表示结果,其基本符号如图 2-15 所示。各连接点表示状态,可取值"0"或"1"。"0"表示某状态不出现,"1"表示某状态出现。

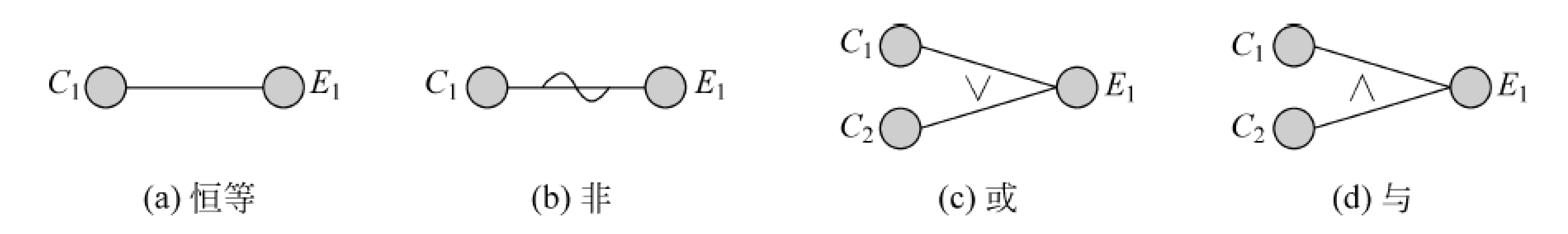


图 2-15 因果关系表示符

主要的原因和结果之间的关系如下:

(1) 恒等。表示原因和结果之间一对一的对应关系。若原因出现,则结果出现;若原因不出现,则结果也不出现。

(2) 非。表示原因和结果之间的一种否定关系。若原因出现,则结果不出现;若原因不出现,反而结果出现。

(3) 或(∨)。表示若几个原因中有一个出现;则结果出现,只有当这几个原因都不出现时,结果才不出现。

(4) 与(∧)。表示若几个原因都出现,则结果才出现;表示若几个原因中有一个不出现,结果就不出现。

3) 表示约束条件的符号

为了表示原因与原因之间、结果与结果之间可能存在的约束条件,在因果图中可以附加一些约束符号。若从输入(原因)考虑,有以下 5 种约束,如图 2-16 所示。

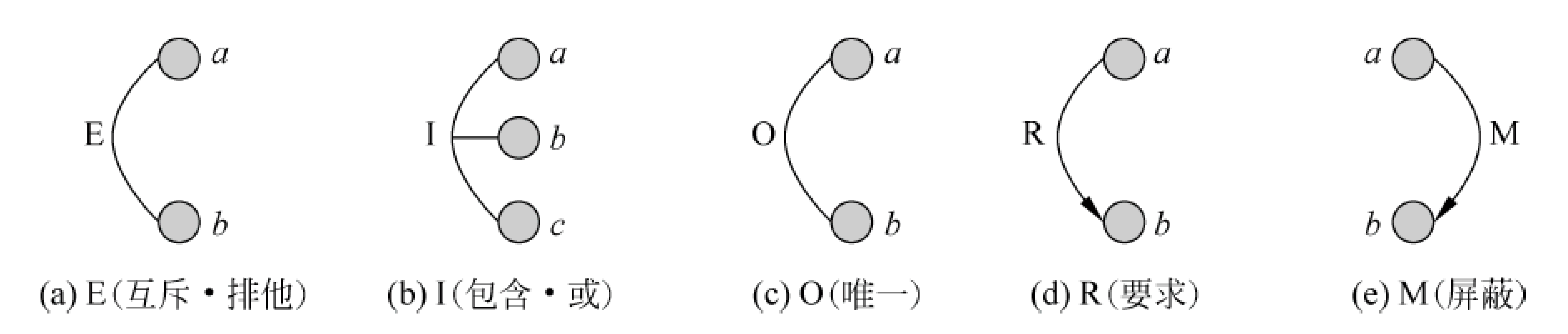


图 2-16 关系约束符号

(1) E(互斥)。表示 a、b 两个原因不会同时成立,两个中最多有一个可能成立。

（2）I(包含)。表示 a、b、c 三个原因中至少有一个必须成立。

（3）O(唯一)。表示 a 和 b 当中必须有一个且仅有一个成立。

（4）R(要求)。表示当 a 出现时，b 必须也出现。不可能 a 出现，b 不出现。

（5）M(屏蔽)。表示 a 是 1 时，b 必须是 0。而当 a 为 0 时，b 的值不定。

5．实例

例 2-5　有一个处理单价为 5 角钱的饮料的自动售货机软件测试用例的设计，其规格说明为：若投入 5 角钱或 1 元钱的硬币，按下“橙汁”或“啤酒”按钮，则相应的饮料就送出来。若售货机没有零钱找，则一个显示“零钱找完”的红灯亮，这时在投入 1 元硬币并按下按钮后，饮料不送出来而且 1 元硬币也退出来；若有零钱找，则显示“零钱找完”的红灯灭，在送出饮料的同时退还 5 角硬币。

（1）分析这一段说明，列出原因和结果，如表 2-25 所示。

表 2-25　例 2-5 说明

原　　因	结　　果
1. 售货机有零钱找	21. 售货机“零钱找完”灯亮
2. 投入 1 元硬币	22. 退还 1 元硬币
3. 投入 5 角硬币	23. 退还 5 角硬币
4. 按下“橙汁”按钮	24. 送出橙汁饮料
5. 按下“啤酒”按钮	25. 送出啤酒饮料

（2）画出因果图，如图 2-17 所示。所有原因结点列在左边，所有结果结点列在右边。建立 4 个中间结点，表示处理的中间状态。

中间结点：11. 投入 1 元硬币且按下饮料按钮。

12. 按下“橙汁”或“啤酒”按钮。

13. 应当找 5 角零钱并且售货机有零钱找。

14. 钱已付清。

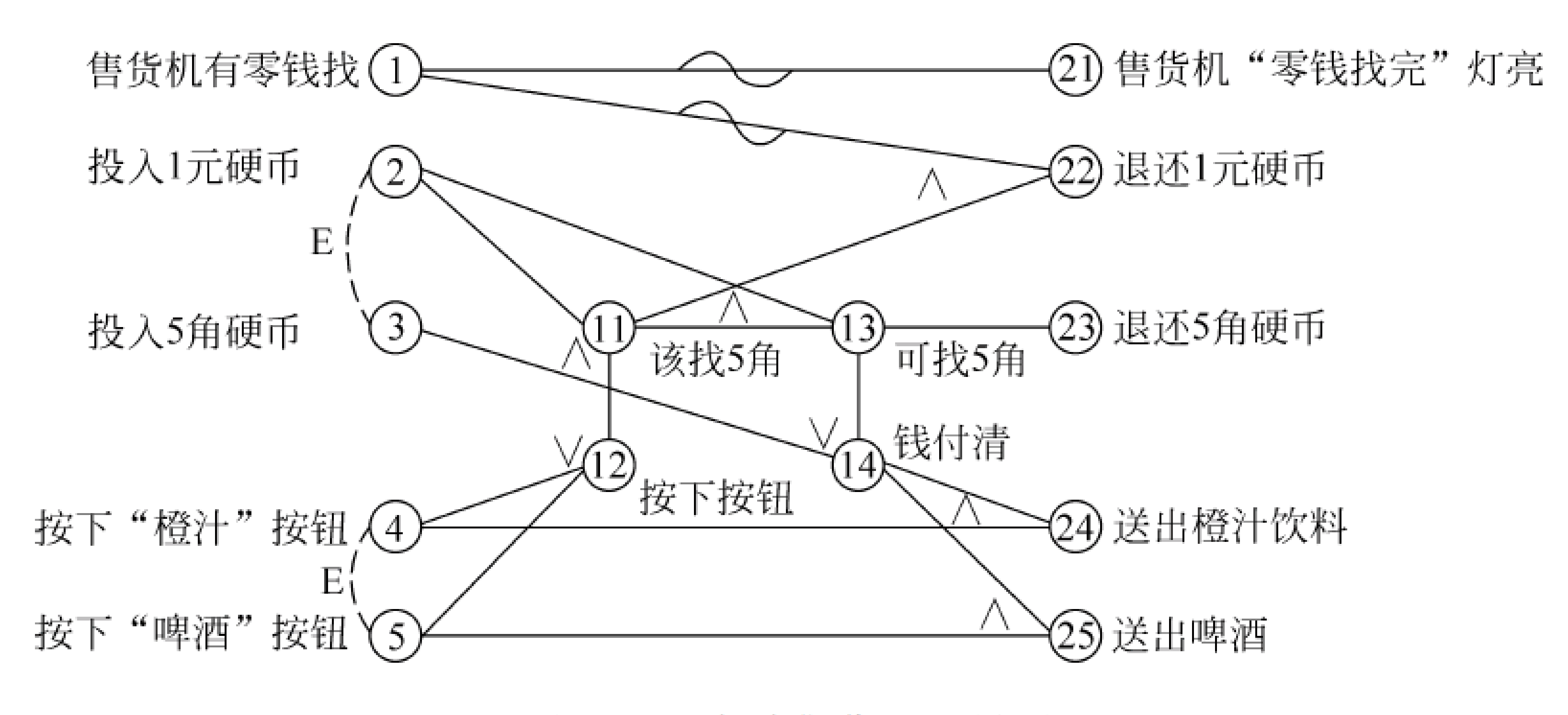


图 2-17　自动售货机因果图

（3）由于原因 2 与 3、4 与 5 不能同时发生，分别加上约束条件 E。

（4）将图 2-17 所示的因果图转换成决策表(如表 2-26 所示)。

表 2-26　自动售货机决策表

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y

6. 优缺点

因果图有助于用一个系统的方法选择出高效的测试用例集，而且它的一个额外好处就是可以指出规格说明的不完整和不明确之处。因果图确实能够产生一组有效的测试用例，但通常它不能够生成全部应该被确定的有效测试用例。在条件的数量和依赖关系增加时，因果图和决策表的规模增加得非常快，从而失去可读性，而且将因果图转换为决策表是最具难度的部分，但目前已经有算法和相应的工具可以支持自动完成。另外，在优化决策表时可能会引入错误，比如忽略了需要考虑的输入和条件的组合。

习题 2.2.3

因果图和决策表的关系是什么？

2.2.4　错误猜测法（Error Guessing Method）

1. 概念

错误猜测法是基于测试人员的经验和直觉来推测系统中可能存在的各种缺陷，有针对性地设计测试用例的方法。这里经验和直觉来自测试人员对待测试软件系统特性的了解和以往在测试工作中的总结。错误猜测法的基本思想是列举出系统中所有可能的缺陷和容易发生缺陷的特殊情况，并根据它们选择测试用例。可以利用不同测试阶段的经验和对被测试软件系统的认识来设计测试用例。例如，用一些非法、错误、不正确和无意义的数据进行输入测试，针对程序代码中的内存分配、内存泄露等问题展开测试等。一般来说，可以从以下几个方面进行错误猜测：软件产品以前版本中存在的问题；受到语言、操作系统、浏览器等环境的限制而可能带来的问题。

2. 原理

常常可以看到这种情况，有些人似乎天生就是干测试的能手。这些人没有用到任何特殊的方法，如因果图、边界值分析等，却似乎有着发现错误的诀窍。对此的一个解释是这些人更多是在下意识中，实践着一种称为错误猜测的测试用例设计技术。接到具体的程序之后，他们利用直觉和经验猜测出错的可能类型，然后编写测试用例来暴露这些错误。

3. 方法

由于错误猜测主要是一项依赖于直觉的非正规的过程，因此很难描述出这种方法的规程。其基本思想是列举可能犯的错误或易发错误情况的清单，然后依据清单来编写测试用例。例如，程序输入中出现 0 这个值就是一种易发的错误。因此，可以编写测试用例，检查特定的输入中有 0，或特定的输出值被强制为 0 的情况。同样，在出现输入输出的数量不定的地方（如某个被搜索列表的条目数量），数量为“没有”和“一个”（对应空列表、仅包含一个条目的列表）也是错误易发生情况。另一个思想是，在阅读规格说明时，联系程序员可能做的假设来确定测试用例（即规格说明中的一些内容会被忽略，要么是由于偶然因素，要么是

程序员认为其显而易见)。

4. 实例

由于无法给出一个规程来,次优的选择就是讨论错误猜测的实质,最好的做法是举出实例。例如测试一个对线性表进行排序的程序,利用错误猜测法要特别测试如下情况:输入的线性表是空的;输入列表仅包含一个条目;输入列表中部分或所有条目的值都相同;输入列表已经按逆序排好;输入的线性表已经排好序。这里列举出的情况可能在程序设计时被忽略。

习题 2.2.4

错误猜测法是否可有可无?

2.2.5 状态转换测试(State Transformation Testing)

1. 概念

状态转换测试,根据测试对象的规格说明,将测试对象抽象为一个状态转换图,根据状态转换图设计测试用例系统检测状态转换过程中可能存在的问题。测试的有效程度取决于状态转换图是否正确反映了测试对象的规格说明。

2. 目标

状态转换测试用于系统检测软件系统中各种状态对于测试对象的功能的影响,测试对象的功能会因为测试对象的状态不同而受影响。已有的测试方法不具备这个特点,它们不能有效检测函数在不同状态下会有不同的行为。

3. 原理

很多情况下,测试对象的输出和行为不仅受当前输入数据的影响,同时还与测试对象之前的执行情况,或者之前的事件或以前的输入数据等有关。为了说明测试对象和历史数据之间的关系引入状态图,状态图是进行状态转换测试设计的基础。

状态转换模型包括状态、转换、事件、动作和它们之间的关系,状态是互不相交的、可辨识的且数量有限的;事件是由输入产生的,可导致状态之间发生转换,转换可以回到其起始状态;转换过程中产生动作或输出。

状态转换测试中,测试对象可以是一个具有不同系统状态的完整系统,也可以是一个面向对象系统中具有不同状态的类。假如由于历史的原因会导致系统不同的表现,就需要用状态转换测试。

4. 方法

设计测试用例执行状态图中的转换。一个测试用例可以执行多个转换。对于每个测试用例要指定:软件的起始状态、软件的输入、预期输出和预期的最终状态。对于该测试用例执行的每个转换要指定以下信息:起点状态、触发状态转换的事件、预期的状态转换发生时

的动作和预期的下一个状态。测试用例可以用来测试软件中有效的转换，也可以测试那些无法推测的转换。

对于状态转换测试，同样可以定义测试强度和完成准则：

(1) 每个状态至少执行一次。

(2) 每个状态转换至少执行一次。

(3) 所有不符合规格说明的状态转换都已检查。

对于一些要求比较高的应用程序，可能还需要声明以下状态转换测试准则：

(1) 所有的状态和输入的组合。

(2) 所有状态转换的组合。

(3) 所有状态的任意顺序的所有转换，也可以是重复的或连续的。

覆盖对象是状态转化模型中的单个转换或多个转换的序列。对于单个转换，转换覆盖率是测试中执行的有效转换占所有转换的百分比，亦称 0-切换覆盖率(0-switch)；对于 n 个转换序列，转换覆盖率是测试中执行的有效 n 转换占所有 n 转换的百分比，亦称 $n-1$-切换覆盖率($n-1$-switch)。状态转换覆盖测试统称 k-切换覆盖(k-switch，$k\geqslant 0$)。

5. 实例

一个时间日期显示软件有 4 个状态模式，即显示时间模式(DISPLAYING TIME (S1))、显示日期模式(DISPLAYING DATE (S2))、设置时间模式(CHANGING TIME(S3))和设置日期模式(CHANGING DATE (S4))，如图 2-18 所示；可以接受 4 个输入，即改变模式('change mode'(CM))、重置('reset'(R))、时间设置('time set'(TS))和日期设置('date set'(DS))。同时，在不同输入条件下，它有 4 个预期输出动作，即显示时间(display time (T))、显示日期(display date (D))、改变时间(alter time (AT))和改变日期(alter date (AD))。

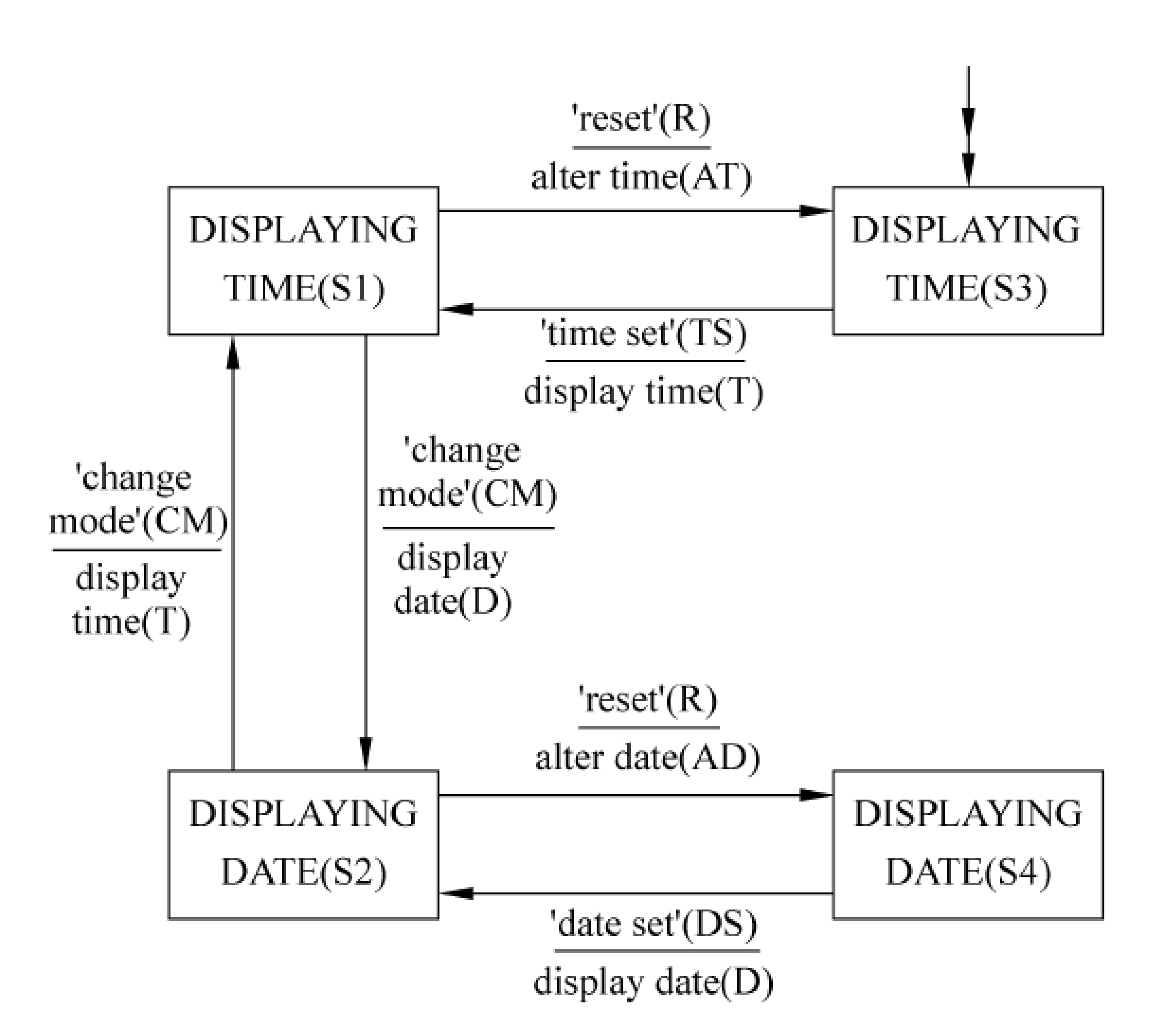


图 2-18 时间/日期软件状态模型

图 2-19 是覆盖所有单个转换的测试用例集合(0-switch 覆盖测试用例集)。例如第一个测试用例起始状态是 S1,输入是 CM,预期输出是 D,终止状态是 S2。这种测试方法可以系统检测各种转换中可能存在的错误,但不能发现转换序列中可能存在的问题。

测试用例	1	2	3	4	5	6
起始状态	S1	S1	S3	S2	S2	S4
输入	CM	R	TS	CM	R	DS
预期输出	D	AT	T	T	AD	D
终止状态	S2	S3	S1	S1	S4	S2

图 2-19 0-switch 覆盖测试用例集

图 2-20 给出了 10 个测试用例,可以覆盖所有两个转换之间的有效组合(1-switch 覆盖测试用例集)。例如,第 1 个测试用例从 S1 出发,输入 CM 进入 S2,再输入 CM,进入 S1 状态。测试的转换序列长度越长,测试覆盖程度就越高,测试就越彻底。

测试用例	1	2	3	4	5	6	7	8	9	10
起始状态	S1	S1	S1	S3	S3	S2	S2	S2	S4	S4
输入	CM	CM	R	TS	TS	CM	CM	R	DS	DS
预期输出	D	D	AT	T	T	T	T	AD	D	D
下个状态	S2	S2	S3	S1	S1	S1	S1	S4	S2	S2
输入	CM	R	TS	CM	R	CM	R	DS	CM	R
预期输出	T	AD	T	D	AT	D	AT	D	T	AD
终止状态	S1	S4	S1	S2	S3	S2	S3	S2	S1	S4

图 2-20 1-switch 覆盖测试用例集

图 2-21 给出的是时间/日期软件的状态表,符号 N 表示无预期输出。图中第 1 列是 4 个状态,第 1 行是 4 个输入,中间每一单元格元素是在相应行状态下对应相应列的输入时,系统的预期输出和最后状态。例如,图 2-21 中右下方最后一个单元格对应在状态 S4,输入是 DS,预期输出是 D 并进入状态 S2。

输入 / 状态	CM	R	TS	DS
S1	S2/D	S3/AT	S1/N	S1/N
S2	S1/T	S4/AD	S2/N	S2/N
S3	S3/N	S3/N	S1/T	S3/N
S4	S4/N	S4/N	S4/N	S2/D

图 2-21 时间/日期软件状态表

一个更为有效的方法是根据状态表逐一设计测试用例(如图 2-22 所示),可以为时间/日期软件生成 16 个测试用例(如图 2-23 所示)。这种设计方法覆盖了系统状态和系统输入之间的全部组合。

状态\输入	CM	R	TS	DS
S1	S2/D（测试用例 1）	S3/AT（测试用例 2）	S1/N（测试用例 3）	S1/N（测试用例 4）
S2	S1/T（测试用例 5）	S4/AD（测试用例 6）	S2/N（测试用例 7）	S2/N（测试用例 8）
S3	S3/N（测试用例 9）	S3/N（测试用例 10）	S1/T（测试用例 11）	S3/N（测试用例 12）
S4	S4/N（测试用例 13）	S4/N（测试用例 14）	S4/N（测试用例 15）	S2/D（测试用例 16）

图 2-22　根据状态表逐一设计测试用例

测试用例	1	2	3	4	5	…	…	…	12	13	14	15	16
起始状态	S1	S1	S1	S1	S2	…	…	…	S3	S4	S4	S4	S4
输入	CM	R	TS	DS	CM	…	…	…	DS	CM	R	TS	DS
预期输出	D	AT	N	N	T	…	…	…	N	N	N	N	D
终止状态	S2	S3	S1	S1	S1	…	…	…	S3	S4	S4	S4	S2

图 2-23　覆盖状态表的测试用例集

6. 优缺点

状态转换测试适用于那些状态起着重要作用的测试对象，测试对象的功能会因为测试对象的状态不同而受影响。已有的测试方法不具备这个特点，它们不能有效检测函数在不同状态下会有不同的行为。

在面向对象系统中，对象可以有不同的状态，操作对象的方法必须能根据不同的状态作出相应的反应，所以状态转换测试技术尤为重要，因为它们考虑面向对象的特征。

当系统的状态数量比较大，状态转换的组合就更大，采用状态转换测试方法，测试用例的数量就会过大，得到足够的覆盖率就不大可能了。

习题 2.2.5

状态转换测试主要检查什么？

2.2.6　语法测试（Syntax Testing）

1. 概念

语法测试是一种不同的测试方法，它基于输入接口的语法变异生成测试用例，因此特别适合在对输入进行确认时发现不足之处。通过语法测试，可以确定合法输入串被接受，非法输入串被拒绝，并且没有输入串会引起系统失效。

2. 目标

语法测试主要针对 3 种类型的错误，即软件不识别正确的串；软件接收不正确的串；软

件在接收一个串时崩溃。

3. 原理

语法测试中的测试用例，也就是软件输入，可以基于软件接口能理解的语言规格说明来创建。自动语法测试需要一个机器可读格式的输入语言形式化描述。如果语言是隐式的，测试人员就必须为语言创建规格说明。规格说明最通用的形式是 BNF 范式和正规表达式，它们都可以用来定义上下文无关文法语言。句子是一个字节序列，按照语言的规则进行排列。上下文无关语言一般用于各种编译器，为输入句子创建语法分析器。上下文无关语言是产生句子的基础，然后把这些句子注入被测软件中，观察软件接收/拒绝这些句子，或是在处理过程中发生失效。

语法测试的有效性取决于建立的语法规格说明是否符合需求。

4. 方法

语法测试一般遵循的原则如下：

(1) 识别目标语言或格式。

(2) 形式化地定义语言语法，例如 BNF 范式。

(3) 通过覆盖输入语言的 BNF 语法图测试正常条件。

(4) 通过执行非法数据测试异常条件。

通过对下面的策略进行相应的改造为语法测试生成测试用例：

(1) 每次产生一个错误，同时保持输入串的其他部分正确。

(2) 一旦为单个错误定义了一组完整的测试用例，那么对两个错误的组合也同样处理，然后三个错误的组合等等。

(3) 每次只关注一个层次，同时尽可能保持更高和更低层次正确。

语法测试没有确定的测试覆盖度量，任何一种覆盖度量都取决于产生有效语法选项的规则和无效语法选项的列表，这些规则和列表都不是十分明确，针对具体语法可能有所不同。

5. 实例

浮点数输入测试，浮点数的 BNF 范式可以表示为：

```
float = int "e" int
int = [" + "|" - "]nat
nat = {dig}
dig = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

其中，int 可有 3 个选择，即 nat[opt_1]，+nat[opt_2]，-nat[opt_3]；nat 有 2 个选择，即单个数字[opt_4]，多个数字[opt_5]；dig 有 10 个数字选择[opt_6]~[opt_15]。因此，可以设计如表 2-27 所示的测试用例覆盖这些可能的选项。

表 2-27　依据语法产生的测试用例

测试用例	浮点数输入	执行的选项	检查结果
1	3e2	opt_1	'valid'
2	+2e+5	opt_2	'valid'
3	−6e−7	opt_3	'valid'
4	6e−2	opt_4	'valid'
5	1234567890e3	opt_5	'valid'
6	0e0	opt_6	'valid'
7	1e1	opt_7	'valid'
8	2e2	opt_8	'valid'
9	3e3	opt_9	'valid'
10	4e4	opt_10	'valid'
11	5e5	opt_11	'valid'
12	6e6	opt_12	'valid'
13	7e7	opt_13	'valid'
14	8e8	opt_14	'valid'
15	9e9	opt_15	'valid'

另一个方面是设计语法的非法测试用例。对语法的 BNF 做变异，主要有以下 4 种变异方式：

- m1　为 BNF 的元素引入一个无效值；
- m2　将 BNF 的元素替换为另一个已定义的值；
- m3　丢失 BNF 的一个元素值；
- m4　增加一个额外元素值。

将这些一般的变异方式应用到具体的语法中，就形成一个具体的变异，例如，对于浮点数的 BNF，每个元素用 el_1、el_2 等表示，原先的 BNF 就可以表示成下面右边的形式：

```
float  =   int "e" int                                     el_1   =  el_2 el_3 el_4
int    =   ["+""-"]nat                                     el_5   =  el_6 el_7
nat    =   {dig}                                           el_8   =  el_9
dig    =   "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|       el_10  =  el_11
```

分别对上面右边的 BNF 做变异，可以产生如表 2-28 所示的测试用例。例如，在表 2-28 中测试用例 1 是对元素 el_2 做变异 m1，即引入一个无效值 x。

表 2-28 依据语法产生的变异测试用例

测试用例	浮点数输入	变 异	元 素	检查结果
1	xeo	m1	x for e1_2	'invalid'
2	oxo	m1	x for e1_3	'invalid'
3	oex	m1	x for e1_4	'invalid'
4	xoeo	m1	x for e1_6	'invalid'
5	＋xeo	m1	x for e1_7	'invalid'
6	eeo	m2	e1_3 for e1_2	'invalid'
7	＋eo	m2	e1_6 for e1_2	'invalid'
8	ooo	m2	e1_2 for e1_3	'invalid'
9	o＋o	m2	e1_6 for e1_3	'invalid'
10	oee	m2	e1_3 for e1_4	'invalid'
11	oe＋	m2	e1_6 for e1_4	'invalid'
12	eoeo	m2	e1_3 for e1_6	'invalid'
13	＋eeo	m2	e1_3 for e1_7	'invalid'
14	＋＋eo	m2	e1_6 for e1_7	'invalid'
15	eo	m3	e1_2	'invalid'
16	oo	m3	e1_3	'invalid'
17	oe	m3	e1_4	'invalid'
18	yoeo	m4	y in e1_1	'invalid'
19	oyeo	m4	y in e1_1	'invalid'
20	oeyo	m4	y in e1_1	'invalid'
21	oeoy	m4	y in e1_1	'invalid'
22	y＋oeo	m4	y in e1_5	'invalid'
23	＋yoeo	m4	y in e1_5	'invalid'
24	＋oyeo	m4	y in e1_5	'invalid'

习题 2.2.6

语法测试主要检查哪些类型的错误？

第3章 开发过程中的测试

软件测试是伴随着软件生命周期每一个阶段的检查和验证活动,包括需求分析、概要设计、详细设计、编码的各个阶段以及维护等。本章着重介绍编码的各个阶段测试,即针对单元模块的测试——单元测试;将各个经过测试的单元模块进行集成的测试——集成测试;系统集成之后进行的系统测试;系统经过维护之后的回归测试;系统出厂前的 α 测试和 β 测试等。每个阶段测试的对象、关注的软件质量问题都有所不同,同时测试的环境和人员也有所不同,但第 2 章介绍的黑盒测试、白盒测试以及静态测试和动态测试等各种测试方法都可以应用到这些测试过程之中。

3.1 单元测试(Unit Testing)

1. 概念

单元测试是在模块源程序代码编写完成之后进行的测试,是集成测试的基础。只有通过了单元测试模块,才可以集成到一起进行集成测试,否则,投入运行的软件就像地基不牢的摩天大楼一样暗藏很多不安全因素。软件开发过程中由于不断地出现需求变更和功能的完善等原因,需要对程序单元的代码进行适当改动,因此单元测试也是一项贯穿于软件开发过程的一种活动。

单元测试是在软件开发过程中进行的最低级别的测试活动,是针对软件设计的最小单位程序模块进行的正确性检验工作。

软件的最小单元可以是一个具体的函数或过程、一个模块、一个类或一个类的方法,它一般具有一些基本的属性,如明确的功能或规格定义、与其他部分明确的接口定义。如果一个单元可以清晰地与同一程序的其他单元划分开来,就可以把它作为软件的一个单元,进行单元测试。因此,一般来说,“单元”是软件里最小的、可以单独执行编码的单位,同时还需具备以下特征:

- 单元必须是可测的;
- 单元的行为或输出是可观测的;
- 有明确的可定义的边界或接口等。

单元测试就是由验证软件单元的实现是否和该单元的说明完全一致的相关联的测试活动组成的。根据软件单元的说明文档(该文档可以是一种说明语言,或一种自然语言,或状

态机等）编写测试用例，对重要的接口、局部数据结构、边界条件、独立路径和错误处理路径，通过代码检查或执行测试用例有效地进行测试。

在结构化编程语言中，一般对函数或子过程进行单元测试；在使用纯C语言的代码中，一般认为一个函数就是一个单元；在可视化环境下或图形用户界面环境下，单元可以是一个窗口或者是这个窗口中相关元素的集合，如一个组合框等；在基于组件开发的环境中，单元可以是一个预先定义的可重用组件，在这种情况下，测试者可以考虑该组件的初始状态成熟度和以前的测试历史，以确定当前执行什么测试；对于Web编程的网页，单元可以是页面上的一个子功能，如一个文字输入窗口或一个功能按钮；在面向对象的语言中，类或类的方法是单元测试的对象。这样可以避免开发人员和测试人员陷入不必要的单元争论中。

2. 目标

单元测试的目标是在于验证代码是否与设计相符合；跟踪需求和设计的实现是否一致，发现设计和需求中存在的错误；发现在编码过程中引入的错误，发现各模块内部的错误，以确保模块内部的一致性与逻辑正确，并使排错工作易于进行。

单元测试（模块测试）是对程序中的单个子程序或过程进行测试的过程，即一开始并不是对整个程序进行测试，而是将注意力集中在对构成程序的较小模块的测试上。这种做法至少有3个好处：第一，模块测试将复杂的程序分而治之，将注意力集中在程序的较小单元上，容易发现程序中的问题，这是人们管理和测试复杂测试对象的一般方法；第二，单元测试可以减轻调试（准确定位并纠正某个已经发现的错误）的难度，因为一旦某个错误被发现出来，就容易知道问题出在什么地方；第三，单元测试可以并行，允许多个被测试单元的测试工作同时展开，可以通过并行工程方法提高测试效率。单元测试的过程包括测试的策略，测试的分析及人员、资源和进度的安排等，单元测试是必须进行的、不能省略的过程。

受过专业训练的独立测试者能比程序员发现更多的错误，但程序员进行单元测试比测试人员更有效，因为测试人员需要更多的时间和精力了解程序。一个独立的测试者可以为开发者提供合理的建议和辅助，以帮助他们计划、组织和执行高效和有效的单元测试，并可为多个程序员提供支持。

单元质量决定系统质量，只有每行代码、每个变量、每个输入数据、每个函数调用的参数和返回值都被测试过，我们才能对软件质量有足够的信心。就像一辆汽车有1～3万个零件，一架波音747飞机有600万个零件，当每个零件可靠性为90%，总体只有34.87%。只有提高每个零件的质量提高到99.99%，才能达到总体99.9%。

在单元测试阶段，某些问题容易被发现且容易修改，但是这些问题如果在后期的测试中被发现，所花的成本将会成倍增长。

3. 原理

通常单元测试考虑以下几个方面的内容：

（1）接口。对单元接口的测试保证进出软件单元的数据流是正确的，这是一个最基本的测试，应该在其他测试之前进行，因为如果数据不能正确地输入和输出，所有的其他测试都是没有实际意义的。

（2）局部数据结构。对单元局部数据结构的测试保证临时存储的数据在算法执行的整

个过程中都能维持其完整性。单元的局部数据结构经常出现错误，一般使用数据定义使用对 Data Definition-use Pair 和程序切片(程序切片是人们从源程序中挑选出来的一个程序子集)测试方法，发现的错误包括：不正确或不一致的类型描述；错误的初始化或默认值；不正确的变量名字；不一致的数据类型；上溢、下溢和地址错误等。除了局部数据结构，全局数据对单元的影响在单元测试过程中也应当进行审查。

(3) 边界条件。保证模块单元在极限或某些严格条件下仍然正确执行，例如，错误往往出现在一个 n 元数组的第 n 个元素被处理的时候，或者一个 i 次循环的第 i 次执行，或允许的最大值或最小值出现的时候。用边界值方法可以有效地发现这类错误。

(4) 独立路径。在控制结构中的所有路径覆盖法或基本路径覆盖法都是需要测试的，以保证在一个模块中的所有语句或分支元素等都能被执行一次。测试用例应当能够发现由于错误计算、不正确的比较或者不正常的控制流而产生的错误，包括：误解的或不正确的算术优先级；混合模式的操作；不正确的初始化；精度不够准确；表达式的不正确符号表示。当比较和控制流紧密地耦合在一起，即控制流的转移是在比较之后发生的，此时测试用例应当能够发现的错误包括：不同数据类型的比较；不正确的逻辑操作或优先级；应该相等的地方由于精度的错误而不能相等；不正确的比较或者变量；不正常的或者不存在的循环终止；当遇到分支循环的时候不能退出；不适当地修改循环变量等。

(5) 错误处理路径。要对所有处理错误的路径进行测试，好的设计要求错误条件是可以预料的，而当错误真的发生的时候，错误处理路径被建立，以重定向或者终止处理。在错误处理部分应当考虑潜在的错误包括：对错误描述不够准确；所报的错误与真正遇到的错误不一致；错误条件在错误处理之前就引起了系统异常；异常条件处理不正确；错误描述没有提供足够的信息来帮助确定错误发生的位置。

4. 方法

在单元测试活动中，软件的每个单元应在与程序的其他部分相隔离的情况下进行测试，一般分两步：人工检查和动态执行跟踪。前者主要保证代码算法的逻辑正确性、清晰性、规范性、一致性和算法高效性，并尽可能地发现程序中潜在的错误；后者通过设计测试用例，执行待测试程序来跟踪和比较实际结果与预期结果，以发现错误。

制定一个正确的单元测试策略至关重要，它能保证单元测试活动的有效进行，目前有自顶向下的单元测试策略、自底向上的单元测试策略和孤立的单元测试策略 3 种方式。

(1) 自顶向下的单元测试策略。先对顶层的单元进行测试，把顶层所调用的单元做成桩模块，其次对第二层进行测试，使用上面已测试的单元做驱动模块，以此类推，直到完成所有模块的测试。这种方法的优点是：在集成测试前提供系统的集成途径，由于详细设计一般都是自顶向下进行设计，这样自顶向下的单元测试策略在执行上同详细设计的顺序一致，该方法可以与详细设计及编码进行重叠操作。但自顶向下的单元测试方法的不足之处是：底层单元的测试须等待顶层单元测试完毕之后才能进行，并行性不好，测试周期长。

(2) 自底向上的单元测试策略。先对模块调用图上最底层的模块进行单元测试，模拟调用该模块的模块做驱动模块；然后再对上面一层做单元测试，用下面已被测试过的模块做桩模块，以此类推，直到测试完所有模块。该方法优点是：在集成测试前提供系统早期的集成途径，不需要桩模块，测试用例可以直接从功能设计中获取，而不必从结构设计中获取，

该方法在详细设计文档缺乏结构细节时比较有效。该方法的不足之处是：无法并行，顶层的单元需要等待底层单元测试完之后在进行测试，测试周期长，不能与详细设计和编码重叠进行。

(3) 孤立的单元测试策略。不考虑模块间的关系，为每个模块设计桩模块和驱动模块，每个模块进行独立的单元测试。其优点是：简单、易操作、可并行。不足之处是：不提供早期的集成途径，需要结构设计信息，使用桩模块和驱动模块。

在单元测试中，影响单元测试质量的因素有：单元测试的人员、单元测试的时间和单元测试的过程。其中单元测试过程包括测试计划制定、测试用例设计与生成、执行单元测试和单元测试的评审。

表 3-1 简单列出了单元测试、集成测试和系统测试的区别和联系，读者可以丰富其内容。

表 3-1 单元测试、集成测试和系统测试的区别与联系

	测试对象	测试内容	测试方法	联系
单元测试	实现具体功能的程序单元	程序逻辑、功能、变量和接口等	白盒测试	单元测试中引入集成测试方法
集成测试	针对概要设计，测试所包含的模块及模块组合	模块间接口、相互关系以及组合后是否达到预期效果	灰盒测试	从测试人员角度，发现软件开发过程中所引入的错误
系统测试	根据需求规格说明书进行，测试软件系统	从用户角度测试系统功能和性能是否满足用户需求	黑盒测试	系统测试从用户角度考虑，单元测试从开发者角度考虑

5. 优缺点

单元测试是一种验证行为，程序中的每一项功能都是通过单元测试来验证它的正确性。它为以后的开发提供支援。就算是开发后期，人们也可以轻松的增加功能或更改程序结构，而不用担心这个过程中会破坏重要的东西。而且它为代码的重构提供了保障。这样，我们就可以更自由地对程序进行改进。

单元测试是一种设计行为，编写单元测试将使测试人员从调用者观察、思考。特别是先写测试(Test-first)，迫使程序员把程序设计成易于调用和可测试的，即迫使程序员解除软件中的耦合。

单元测试是一种编写文档的行为，单元测试产生的文档是一种无价的文档，它是展示函数或类如何使用的最佳文档。这份文档是可编译、可运行的，并且它保持最新，永远与代码同步。

单元测试具有回归性，即模块单元需要不断地改进和再验证，自动化的单元测试可以支持模块单元代码的回归测试，编写完成之后，可以随时随地地快速运行测试。

但单元测试毕竟是对局部模块的测试，无法保证当这些模块集成在一起时，系统是否工作，这就需要进行集成测试。

习题 3.1

1. 简述单元测试的主要目标。

2. 单元测试的优缺点有哪些?

3. 单元测试与集成测试、系统测试有哪些区别和联系?

3.2 集成测试(Integration Testing)

尽管每个模块都经过了严格的单元测试,每个模块都能单独工作,但将这些模块集成在一起之后,往往可能出现异常情况,例如某些模块不能正常工作、接口数据丢失、模块之间的不良影响、误差的不断积累等。因此,单元测试无法代替集成测试,每个模块的性能最优并不能保证集成之后的指标最优,在单元测试之后,必须进行系统的集成测试。

1. 概念

集成测试是在单元测试的基础上,将所有已通过单元测试的模块,采用适当的集成策略,按照概要设计的要求组装为子系统或系统,目的是确保各单元模块组合在一起后能够按照概要设计要求运行,检测增量的行为是否正确。集成测试之前必须进行单元测试,否则其效果和效率必将大打折扣。集成测试也叫组装测试、联合测试、子系统测试或部件测试,测试用例的设计方法可以使用黑盒测试和白盒测试,因此人们一般把集成测试归为灰盒测试。

2. 目标

集成测试主要检测模块之间的接口和各个模块集成后实现的功能,具体包括以下几个方面的问题:

(1) 几个模块连接起来后,穿过模块接口的数据是否会丢失,是否能够按照预期将数据传递给另外一个模块。

(2) 各个模块连接起来后,需要检测是否存在在单元测试时没有发现的资源竞争问题。

(3) 将通过单元测试的子功能模块集成到一起,检测是否可以实现所期望的软件功能。

(4) 兼容性。检查当引入一个模块后,该模块是否对其他与之相关的模块产生负面的影响。

(5) 全局数据结构是否正确,是否被不恰当地修改了。

(6) 在集成后,每个模块的误差是否会扩大,是否会达到不可接受的程度。

3. 原理

由于集成测试处在单元测试和系统测试的过渡阶段,因此集成测试工作的好坏在整个测试过程中起着至关重要的作用,它负责查找各个模块集成前,在单元测试时遗漏的以及无法发现的问题,而且为系统测试的进行打基础。若集成测试工作做得不够充分和周密,会直接降低系统测试的效果和效率,例如,难以对系统测试时发现的错误进行定位,浪费宝贵的测试资源。

与软件开发之前要做好系统分析一样,集成测试也要进行系统的分析,具体包括体系结构分析、模块分析、接口分析、风险分析、可测试性分析和集成测试策略分析等。

4. 方法

周密细致的集成测试分析为选择和确定集成测试方法提供了重要的参考依据,因此,可以说集成测试方法是建立在测试分析基础上的,具体包括:大爆炸集成、自顶向下集成、自底向上集成、两头夹集成、基于风险的集成、基于事件的集成、基于使用的集成等,以下简介这些方法。

(1) 大爆炸集成。也称为一次性组装或整体拼装,就是把所有通过单元测试的模块一次性集成到一起进行测试,不考虑组件间的互相依赖及可能存在的风险,这样尽可能缩短测试时间,使用最少的测试用例验证系统。

(2) 自顶向下集成。就是按照系统层次结构图,以主程序模块为中心,自上而下按照深度优先或者广度优先策略,对每个模块一边组装一边进行测试。

(3) 自底向上集成。从依赖性最小的底层模块开始,按照层次结构图,逐层向上集成,对于某一层次的特定模块,因为它的子模块已经组装并测试完成,所以不再需要桩模块,在测试过程中如果需要从子模块得到信息可以直接运行子模块得到,即在测试过程中只需要开发相应的驱动模块。

(4) 两头夹集成。也称三明治集成,是一种混合增量式测试策略,综合了自顶向下和自底向上两种集成方法的特点。该方法首先在软件层次结构图中选定一层,确定该层以下部分进行自底向上集成测试,该层以上部分进行自顶向下集成测试,最后对系统进行整体测试。

(5) 基于功能的集成。从功能实现的角度出发,按照模块的功能重要程度组织模块的集成顺序,先对开发中最主要的功能模块进行集成测试,以此类推最后完成整个系统的集成测试。

(6) 基于风险的集成。在软件系统中,风险最高的模块间的集成往往是错误集中的地方,因此尽早地验证这些接口有助于系统的稳定。基于风险的集成和基于功能的集成有类似的地方,两者可以结合使用。

(7) 基于使用的集成。针对面向对象系统,根据类之间的依赖关系来集成系统,从而验证系统的稳定性。

还有一些其他方法,例如基于调用图的集成、基于路径的集成、基于进度的集成、基于事件的集成等,这些方法都是从一些不同角度和不同方面提出集成策略,可以有效弥补已有方法的不足,是集成测试策略的重要补充,具有各自的合理性和科学价值。

5. 优缺点

集成测试是一个由单元到系统的过渡性测试,由于其位置的特殊性,而容易被忽视。集成测试可以保证各个经过测试的单个模块在集成起来之后也能正常工作,但集成测试的效果往往依赖于集成策略。

习题 3.2

1. 集成测试与软件开发具有什么样的关系?
2. 集成测试策略一共有多少种?简述它们的特点。

3.3　系统测试(System Testing)

所有单元经集成测试之后,各模块被连接起来,形成相对完整的系统,此时模块内部及模块之间的接口在现有测试条件下,不再发现错误,测试进入系统测试阶段。系统测试属于黑盒测试,包括功能测试、性能测试等多个不同的测试组成。

1. 概念

系统测试是将已经过良好的集成测试的软件系统,作为整个计算机系统的一部分,与计算机硬件、外部设备、支持软件、数据以及人员等其他系统元素结合在一起,在实际使用环境下对计算机系统进行一系列的严格测试,以发现软件中潜在的缺陷,保证系统交付给用户之后能够正常使用,满足用户需求。系统测试是产品交付前最后一个测试环节,具有重要的地位。

2. 目标

系统测试的目标是保证开发方交给用户的软件产品能够满足用户的需求,已经集成的系统中每个部分都可以正确地完成指定的功能,因此,系统测试应在实际的用户环境下执行,一般涉及软件、硬件、网络等多方面因素,除了要进行功能测试之外,还要对软件性能、安全性、兼容性、用户界面、可恢复性、可安装性等方面质量属性进行有针对性的测试。

3. 原理

系统测试要通过与系统的需求定义做比较,发现软件与系统定义不符合或与之矛盾的地方,以验证软件系统的功能和性能等满足其规约所指定的要求。系统测试的测试用例应根据需求分析说明书来设计,并在实际使用环境下来运行。

由于软件只是计算机系统中的一个组成部分,软件开发完成以后,最终还要与系统中其他部分配套运行。系统在投入运行以前各部分需完成组装和确认测试,以保证各组成部分能够协调运行,正常工作。

4. 方法

与其他测试一样,系统测试也应该按照测试计划进行,其输入、输出和其他动态行为应该与软件规约进行对比。软件系统测试的方法很多,主要有功能测试、性能测试、随机测试、安全性测试、容量测试、压力测试、兼容性测试、转换测试、可用性测试、文档测试、备份测试、恢复性测试、安装测试、在线帮助测试、用户界面测试和协议一致性测试等,具体这些方面的测试方法将在下一章详细介绍。系统测试执行中一个关键的考虑是不能由程序员来进行测试,最好必须选择不受开发方左右的人群,或第三方机构来进行系统测试。

习题 3.3

1. 简述系统测试定义。

2. 系统测试应该由什么人进行？
3. 系统测试的主要测试技术有哪些？

3.4 验收测试（Acceptance Testing）

软件产品在开发完成之后，要进行软件的验收和交付。验收测试在系统测试通过后开始，在某种意义上是由用户或客户进行的系统测试，但又不是系统测试的重复，验收测试不同于系统测试的主要之处是，验收测试以用户和用户代表为主，以测试人员为辅的测试，一般在用户方的真实使用环境中进行，使用真实的数据。

1. 概念

验收测试是根据用户的需求而进行的，是整个测试过程中的最后阶段。在执行这类测试时，最终用户要参与其中。验收测试计划过程应该在需求确定之后尽快开始进行，在软件生命周期的早期进行此项工作时很重要，因为验收测试的出口准则为产品的验收奠定了基础，验收测试计划和验收测试用例应该非常准确地描述未来软件产品的特性，在外包开发的情况下，验收测试计划和测试用例一般要成为外包合同的一个组成部分。

2. 目标

验收测试是证明需求的有效性和为取得用户的认可而提供支持的一种很有价值的手段。验收测试和系统测试的主要差别是测试的主体，即由谁进行的测试工作。当用户在系统测试中起到十分积极的作用时，而且测试环境足够真实，那么验收测试和系统测试合并在一起是非常有意义的。另外一种情况，用户不参与任何测试，验收测试由专门的测试团队独立做，此时验收测试和系统测试是一样的。

3. 原理

用户或代表用户的第三方测试机构在验收测试过程中起着十分重要的作用，他们需要根据需求，识别业务风险，建立、更新或评审验收测试计划，定义真实的基于场景的测试，提供真实的测试数据，执行测试，评审测试输出，以及提供验收标准等工作。

在验收测试中需求的跟踪是重要方面，需求跟踪就是指确保一个或多个测试用例能够覆盖到每个需求，需求跟踪是一个较高等级的覆盖度量，这个度量也是测试有效性度量的一种手段，验收测试需要向用户或其代表表明所有的需求都得到了满足。

4. 方法

验收测试的方法与系统测试的方法是一致的，只是测试主体不一样。通常验收测试具有 α 测试和 β 测试两种不同类型。后面章节将单独介绍。

经过集成测试，已经按照设计把所有模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是验收测试的任务。但是，什么样的软件才是有效的呢？软件有效性的一个简单定义是：如果软件的功能和性能如同用户所合理地期待的那样，则软件是有效的。在需求分析阶段产生的文档准确地描述了用户对软

件的合理期望，因此是软件有效的标准，也是验收测试的基础。

验收测试的目的是向未来的用户表明系统能够像预定要求那样工作。验收测试的范围与系统测试类似，但是也有一些差别，具体如下：

(1) 某些已经测试过的纯粹技术性的特点可能不需要再次测试。

(2) 对用户特别感兴趣的功能或性能，可能需要增加一些测试。

(3) 通常主要使用生产中的实际数据进行测试。

(4) 可能需要设计并执行一些与用户使用步骤有关的测试。

验收测试必须有用户积极参与，或者以用户为主进行。用户应该参加设计测试方案，使用用户接口输入测试数据并且分析评价测试的输出结果。为了使用户能够积极主动地参与验收测试，特别是为了使用户能有效地使用这个系统，通常在验收之前由开发部门对用户进行培训。

验收测试一般使用黑盒测试法。应该仔细设计测试计划和测试过程，测试计划包括要进行的测试的种类和进度安排，测试过程规定用来检验软件是否与需求一致的测试方案。通过测试要保证软件能满足所有功能要求，能达到每个性能要求，文档资料是准确而完整的，此外，还应该保证软件能满足其他预定的要求(例如，可移植性、兼容性和可维护性等等)。

验收测试有下列两种可能的结果：

(1) 功能和性能与用户要求一致，软件是可以接受的。

(2) 功能或性能与用户的要求有差距。

在这个测试阶段发现的问题往往和需求分析阶段的差错有关，涉及的面通常比较广，因此解决起来也比较困难。为了确定解决验收测试过程中发现的软件缺陷或错误的策略，通常需要和用户充分协商。

验收测试的一个重要内容是复查软件配置。复查的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，文档与程序一致，具有维护阶段所必需的细节，而且已经编排好目录。

除了按合同规定的内容和要求由人工审查软件配置之外，在验收测试的过程中应该严格遵循用户指南以及其他操作程序，以便检验这些使用手册的完整性和正确性。必须仔细记录发现的遗漏或错误，并且适当地补充和改正。

习题 3.4

1. 验收测试主要检查哪些方面的内容？
2. 怎样进行验收测试？

3.5 回归测试(Regression Testing)

软件开发不是一次性完成的，需要经过不断的改进、演化和升级，这些改变会给软件系统带来风险：一方面，软件系统的演化和升级是否正确；另一方面，关于软件的修改是否可能会引入新的错误。在软件生命周期中的任何一个阶段，只要软件发生了改变，就有可能给

该软件带来问题,而回归测试是一种验证已变更的系统的完整性和正确性的测试技术。因此,每当软件发生变化时,就必须重新测试,以便确定修改是否达到了预期目的,检查修改是否损害了原有的正常功能。同时还要补充新的测试用例来测试新的或被修改的功能。为了验证修改的正确性及其影响就必须进行回归测试。

1. 概念

回归测试是指待测试软件中的错误被修改以后或增加了新的功能等,重新进行测试以确认修改的正确性,且没有引入新的错误或导致其他代码产生错误。回归测试的自动化将大幅降低系统测试、维护升级等阶段的成本。

回归测试作为软件生命周期的一个组成部分,在整个软件测试过程中占有很大的工作量比重,软件开发的各个阶段都会进行多次回归测试。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行的更加频繁,而在极限编程方法中,更是要求每天都进行若干次回归测试。因此,通过选择正确的回归测试策略来改进回归测试的效率和有效性是非常有意义的。

2. 方法

回归测试需要时间、经费和人力来计划、实施和管理。为了在给定的预算和进度下尽可能有效率和有效地进行回归测试,需要对测试用例集进行维护并依据一定的策略选择相应的回归测试用例,包括删除过时的测试用例、改进不受控制的测试用例、删除冗余的测试用例、增添新的测试用例等,然后再测试全部用例。测试用例选择策略包括基于风险选择测试、基于操作剖面选择测试等。有了测试用例集的维护方法和回归测试包的选择策略,回归测试可遵循下述基本过程进行:

(1) 识别出软件中被修改的部分。

(2) 从原基线测试用例集 T 中,排除所有不再适用的测试用例,确定那些对新的软件版本依然有效的测试用例,其结果是建立一个新的基线测试用例集 T0。

(3) 依据一定的策略从 T0 中选择测试用例测试被修改的软件。

(4) 如果必要,生成新的测试用例集 T1,用于测试 T0 无法充分测试的软件部分。

(5) 用 T1 执行修改后的软件。

其中第(2)和第(3)步测试验证修改是否破坏了现有的功能,第(4)和第(5)步测试验证修改工作本身。

回归测试的另一个重要问题是回归测试用例集的优化,即把回归测试用例按照执行先后进行排序,使得按照该次序执行时,回归测试可以最早发现软件故障,或最快达到覆盖要求,或在测试资源有限的情况下,只执行最重要的测试用例。

习题 3.5

1. 为什么要进行回归测试?
2. 回归测试与一般的软件测试不同点在哪里?
3. 回归测试中有哪些关键问题?

3.6 α测试

1. 概念

α测试是在开发机构内部的用户在模拟实际使用环境下进行的测试，通常开发者就在用户旁边，随时记下错误情况和使用中的问题，这一种在受控制的环境下进行的验收测试。α测试前应当将测试目的明确地传达给测试参与人员，应当向测试参与人员介绍一些项目的历史背景知识，测试人员要在测试期间提供协助，并给出测试的一般规则。

2. 目标

α测试主要用于发现以下一些问题：概念性缺陷或者与主题不协调的地方；发现与功能需求和项目规格不符合的地方；发现拼写、标点以及习惯用法方面的错误；发现图形的位置错误；发现不准确、不清晰或者不完整的图形及标题等。

3. 方法

参与α测试的人员需要随时记下对系统的建议，建议应该足够详细，以便能指导修改；要按照计划进行α测试，在时间不足的情况下，可以提醒测试参与人员关注软件系统的重要部分，对系统提出必须修改、一般修改或改进型修改等建议。

进行α测试时可能会使用到观察实验室，利用摄像机等工具记录测试现场。

4. 优缺点

α测试是由一个用户在开发环境下进行的测试，也可以是开发机构内部的用户在模拟实际操作环境下进行的测试。α测试的目的是评价软件产品的FLURPS，即功能(Function)、局域化(Localization)、可使用性(Usability)、可靠性(Reliability)、性能(Performance)和支持(Support)，尤其注重产品的界面和特色。α测试人员是除产品开发人员之外首先见到产品的人，他们提出的功能和修改意见是特别有价值的。α测试可以从产品编码结束之时开始，或在单元测试完成之后开始，也可在验收测试过程中产品达到一定的稳定和可靠程度以后再开始。但α测试不是在真实的用户环境中的测试，有些在用户环境中才出现的问题很难检测到。

习题 3.6

α测试主要检查软件哪些方面的问题？

3.7 β测试

1. 概念

β测试是由软件的多个用户在实际使用环境下进行的测试，这些测试参与人员是与公

司签订了 β 测试合同的外部用户，他们被要求使用软件系统，并愿意返回有关错误信息给开发公司，与 α 测试不同是，开发者不在测试现场，因而，β 测试是在开发者无法控制的环境下进行的软件测试。

2. 方法

经过 α 测试调整的软件产品称为 β 版本。β 测试是由软件的多个用户在实际使用环境下进行的测试，这些用户返回有关错误信息给开发者。测试时，开发者通常不在测试现场。因而，β 测试是在开发者无法控制的环境下进行的软件现场应用。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告。β 测试主要衡量产品的 FLURPS，着重于产品的支持性，包括文档、客户培训和支持产品生产能力。

只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。它处在整个测试的最后阶段。同时，产品的所有手册文本也应该在此阶段完全定稿。

3. 优缺点

β 测试通常在产品发布到市场之前，邀请公司的客户参与产品的测试工作，这种方法可以有效提升产品的价值，因为它使那些实际的客户有机会把自己的意见渗透到公司产品的设计、功能和使用过程中。

尽管 β 测试越来越广泛，但它并不能代替 α 测试，因为 β 测试人员不是专业的测试人员，很难发现深层次的问题，更多的发现是停留在易用性方面。其次，β 测试是在不可控的环境下进行的，因此无法了解 β 测试人员实际是如何操作系统的，很多 β 测试人员反馈的问题是由于使用不当引起的，而且 β 测试人员反馈的问题信息很简单，经常无法重现。

习题 3.7

α 测试与 β 测试有什么不同？

3.8 γ 测试

γ 测试是在 α 测试与 β 测试之后的第 3 个阶段，此时产品已经相当成熟，只需在个别地方，如用户手册、软件包等再做进一步的优化处理即可上市发行。

习题 3.8

伴随软件开发整个过程，有哪些主要的阶段性测试？

第4章 软件特性及方面的测试

软件由很多部分组成，如文档、程序以及支持服务等；软件具有很多功能，如安装、卸载以及使用等；衡量软件质量需要考虑很多因素及各种性能，例如，安全性、容错性、可靠性等。因此可以根据需要，对软件的每个部分、每个局部功能以及每个质量属性进行专门的测试，本章重点介绍软件的各种专门特性以及各个方面的测试。

4.1 压力测试(Stress Testing)

1. 概念

压力测试(Stress Testing)，也称强度测试，是指持续不断地给被测试系统增加压力，直到被测试系统被压垮，从而确定系统能承受的最大压力。它属于性能测试，但与常规的性能测试不同，因为常规的性能测试是指在正常的软硬件环境下运行，不向其施加任何压力的性能。压力测试为了发现在什么条件下系统性能变得不可承受，以获得系统能够提供的最大服务级别。

一个好的软件系统在极限状态下，即使来不及响应，也不会死机，并能在负荷恢复正常后一段时间内可以恢复正常运转。如果一个软件系统在极限压力下会完全死机，并且不得不重启系统才能正常工作，人们对此往往不会满意。当然，如果一个系统只要重新启动，就能恢复正常工作，有时也算一个优点。

压力测试中存在一个累积效应的问题，在压力测试过程中为了确保"干净"的测试环境，需要频繁地让系统重启，这样有利于分析问题。但这种方式容易忽略累积效应，使得一些缺陷无法及时发现。特别是服务器端的压力测试，往往单次测试时问题表现不明显，只有日积月累才能造成严重问题。

2. 目标

压力测试强制软件运行在超出极限的不正常情况下，检验软件可以运行到何种程度，具体测试要求包括：提供最大处理的信息量；提供数据能力的饱和实验指标；提供最大存储范围(如常驻内存、缓冲、表格区、临时信息区)；在能力降级时进行测试；进行其他健壮性测试(测试在人为错误下的反应，如寄存器数据的跳变、错误的接口状态)；进行持续规定时间且连续不能中断的测试。

压力测试应该测试所有产品文档中声明的程序行为的所有极限值。应该检查软件能处

理的文件大小、能驱动的打印机数量，以及能管理的终端、调制解调器和内存数量，打开程序允许的最大数量文件。还应该检查程序能及时处理多少事件，一段时间内又能处理多少。如果没有规定极限，就测试一个非常大的值，看看程序如何处理，如果无法处理，就作为缺陷。

3. 方法

在进行压力测试时，首先要找到影响软件系统的负载因素，例如，输入输出的数据流大小、外部中断的个数与频率等。然后要确定各个因素的影响范围，即负载的大小，如外部中断这个负载因素直接影响到软件的实时响应特性。其次设计测试用例使其对系统产生极端影响，如设计一组负载值，由小到大变化，直到能够观察到系统的非正常状态出现。最后，降低负载强度，考察系统的恢复情况，即沿着设计的负载值，由大到小变化，观察系统是否可以恢复正常的运行状态。

总的来讲，压力测试主要考虑下列 3 个方面：

(1) 长时间测试对软件造成的影响，主要检查软件在数据存储、数据计算精度等方面的性能。

(2) 大数据量对软件的影响，主要检查软件中数据库设计及出错处理和异常设计的合理性。

(3) 高频率数据对软件的影响，主要检查带中断处理的软件中对中断优先级的设置、中断服务例程的处理机制是否合理有效。

习题 4.1

压力测试主要检查软件哪些方面的能力?

4.2 负载测试(Load Testing)

1. 概念

负载测试与压力测试十分相似，通常是让被测试系统在其能忍受的压力极限范围内或临界状态下连续运行，来测试系统的稳定性，目的是找到系统的处理极限，为系统调优提供依据。负载测试与压力测试的区别在于负载测试侧重于压力持续的时间，而压力测试则更加强调施加压力的大小。

负载测试是给系统一个特定的条件，观察系统的反应情况的一个过程。具体说来，负载测试是对于分析出来的系统的指标，在不同的负载级别上逐步加大或减小以确定系统的反应结果。系统的反应结果一般包括事务处理成功率、处理速度、系统总体响应时间、用户 CPU 负载、用户内存使用等等。一般来说，由于其测试的复杂性及繁琐性，负载测试是借助工具在模拟的环境中实施的。

对于压力测试的定义是，将负载测试中分析的指标赋予一个系统容纳范围之外的数值，使其对于系统来说成为一种压力，在这样的条件下观察系统的反应以及各个参数的变化。它的目的是测试系统所能承受的最大范围以及系统的鲁棒性，有时也会观察系统在经受高强度压力后的恢复是否良好。压力测试可能会模拟大量的用户同时访问站点，或者模拟某

些用户频繁地访问站点等方式。高强度的测试因系统的不同而持续的时间有差别，区别可能在数十秒到几天之差。在压力测试这样高强度、极端条件的测试下，人们可能发现系统在设计的时候难以发现或者未经考虑的情况，也会出现各种难以预知的结果，因为在这样苛刻的条件下很可能测试到一些平时根本不会测试到的情形。对于各种莫名其妙的结果，需要精准、无遗漏的记录和分析，并考虑有无改进的必要，进而加以改进。

一般来说，负载测试是优先于压力测试的。对于一个系统来说往往需要先满足其正常指标范围内的要求，然后再去考虑极端情况。而且，压力测试所需要的压力区间需要由负载测试来测定或明确。

性能测试是查看系统在一定负载下的运行状态，并由此分析其瓶颈。其关键的部分在于为系统提供可控制的负载。对于这样的负载，组织大量的用户进行真正的测试是不现实的。因为即使通过 β 测试这样的方法找到了大量的用户，也没法要求这些用户按照既定方针有序地实行。并且，一旦出现了问题，就需要将这样大规模的复杂的测试进行更多的次数，这将是一件成本十分巨大的事，要想重现同样的情景，也是十分困难的。因此，提供可控制的负载一般都是由测试人员模拟进行的。现今的公司对于性能测试的重要性认识在逐渐提高，但是往往理解趋于偏颇。很多测试员会把“负载”这个概念单纯地理解为一个变量的变化。例如，1000 个用户同时访问、1 个用户访问 1000 次等等。

2. 方法

总体来说，为系统提供压力一般通过重复、并发、量级这些方法。还有一种非常有必要却经常被忽视的方法，就是随机变化。

(1) 重复。重复是性能测试中最为基本的手段。重复很容易理解，就是将同样的行为反复执行。性能测试将确定在多次重复中系统能否工作正常，每次请求能否得到合理有效的满足。任何产品都不会由客户一次性使用，因此重复测试是非常必要的。如果用户在对一个站点进行若干次同样内容的访问后出现了问题，势必引起用户的疑问和不满。但是，结合程序的性质来说，仅仅重复是不可能单独地解决问题的，因此这一操作一定是和其他途径结合起来。

(2) 并发。并发是同时执行多个操作的行为，其思路类似于动员大量人力同时进行某些活动。执行的多个操作，可能相同，也可能不同，将因测试内容而异。并发并非适用于所有的系统，但是绝大部分系统，特别是面向广域网的系统，都是要满足并行执行这一基本条件的。例如，Web 性能测试可能模拟很多用户机，它们同时对一些共享数据进行访问。这时候系统对共享数据访问处理就显得至关重要了。现实中的系统对共享数据的处理往往趋于复杂，必须保证在复杂的操作中不会出现不同步、死锁、闲置时间过长等问题的出现。然而，并发的问题出现往往是由某个线程被其他线程中断带来的，而这种情况的出现带有偶然性(后运行的进程 2 会切断先前的进程 1，但是如果让进程 2 先运行，进程 1 不会切断它，这种情况经常存在)。因此，并发往往是和重复结合在一起的。

(3) 量级。对于用户的每一次请求，除了请求过多、请求过于频繁可能给系统带来负担之外，还有一个带来负担的情况就是请求过于复杂。例如，用户向网页输入一条字符串，谁也无法保证用户会输入什么样的字符串，可能特别长，可能带有无法预料的字符等等。或者对于用户传送来的一个数据，其大小、精度等要求和输入的速度和频度都会增加操作的复杂

性，换言之，就是增加了操作的量级。有些时候，量级是不需要考虑的，比如呈现一个选择列表供用户选择，那么输入结果非此即彼。但是用户所输入的信息不可能全部做成选择的形式：最简单的例子，用户输入自己的将要创建的用户名，做成选择的形式完全不现实。这时候，就需要在测试的时候，加大测试内容的量级了。测试应该模拟出各种各样的复杂输入，结合其他两个原则，发现潜在的问题。

(4) 随机变化。为了使这种庞大而费时费力的测试更加有效率，每个测试都应该具有随机性质。在每次测试中都尽可能地探索程序的更多路径。随着现今系统的复杂度越来越高，想要穷尽地测试完系统的每一种路径组合，那可能是一个只能在理论上实现的情形。比如，重复的时候，控制时间间隔或者操作次数或者访问顺序的变化；使用并发的时候，更改同时访问的数目或所访问的内容；使用量级的时候，更改每次发送的数值等等。需要指出的是，测试如果处于完全随机的状态，可能难以回复到某次得到异常结果的情形。如果对于不同的变量使用同样的随机种子，这样发现错误的概率要更大一些。

3. 负载测试常见的性能指标

负载测试所针对的系统往往是现实中需要投入市场的系统，以下这些性能指标往往需要重点注意，包括响应时间、并发量、吞吐量、用户放弃率、用户行为等。

(1) 响应时间。响应时间即用户等待的时间，具体来说，就是从用户发送请求开始，到接受到系统反馈信息的整个等待过程。这个时间往往是和并发访问的客户数量相关的，用户数目多的时候，往往相应时间较长，反之则较短。在早期的互联网开发中，这个指标的重要性往往不如其他的重要，由于硬件等条件的限制，那个时代用户以得到准确良好的数据为前提，对于较慢的反应速度有良好的容忍率。然而，随着互联网技术的飞速发展以及不断完善，用户早已不再满足于简单地得到数据，他们往往更希望数据得到的又快速又准确。进一步地，随着互联网功能的拓展，用户在页面上与服务器的互动行为将更加频繁。这些现象都意味着响应时间的重要性逐渐上升。据 Zona Research 的调查，超过 80%的用户在出现多于 8 秒的延迟的情况就会表示不满(即所谓的“8 秒钟准则”)，而其中相当大的人群会选择更换网站，这样的问题积累多了会变得十分严重：每年由于下载速度令人难以容忍的情况导致的损失高达十几亿美元。

(2) 并发量。并发量即同时对网站进行并发访问的用户数量。而对于并发量指标，目前学术界并未达成共识，这是因为系统之间往往区别很大，其配置、复杂程度、用户数量及频率都随着不同要求的变化而变化，所以并发量应结合现实的开发情况和环境而设计，而不是越高越好。并发量的计算方法是：首先估计系统的用户数量，再按照它们使用的频率估计并发比例，它们的乘积就是并发量。并发量要想更加精确，需要对该系统中不同种类的用户进行分别的频率计算。这个分类、计算过程涉及具体专业的不同知识和各自的经验公式，本书不再详述。

(3) 吞吐量。吞吐量是指在某一个特定的时间单位内 Web 应用所处理的用户请求数目。一般来说，用户的请求数目越多，吞吐量就会越大。并发处理的用户数目达到最大值的时候系统的性能往往是人们所需要注意的，因为这里往往是系统的瓶颈。

(4) 用户放弃率。用户放弃率是一个结合了其他指标的综合性指标，它指的是测试过程中用户将因为网页的错误、响应缓慢而无法忍受最后放弃访问网页的情况。在性能测试

的时候，这个比率需要考虑进去，否则，可能会出现设计的负载量脱离实际情况的时候。事实上，当测试量过于庞大的时候，往往就进入了压力测试的范围，因此，这个比率在界定系统指标的时候需要考虑。

（5）用户行为。用户的行为往往不会像简单的测试所进行的那样，例如，重复地发送同样的信息。相反，用户的行为往往是非常复杂的，也是非常难以预料的。例如，对网页的熟悉程度往往决定了用户的行为的不同，比如如果一个网页的用户多为老用户（如专业性很强的论坛），那么其操作可能普遍比较迅速，对于这样的网页的测试，应该和那些更多面向新用户的网页（如新闻网站）有所区别。网页的内容也可能成为决定用户行为的因素，比方说，网页游戏和视频网站的操作肯定是不一样的，前者可能要面对非常多的鼠标点击，后者则要少得多。再如，以用户输入为主的页面（如博客），和以提供信息为主的页面（如新闻），其用户行为也是显著不同的。在设计负载的过程中，用户行为也要酌情考虑。Web 性能测试的指标不仅仅局限于以上这些，页面请求次数、页面请求分布、用户会话等指标都可能在不同的网站被重点关注，并且，不同的系统所需要关注的也经常很不一样，比如，聊天网站对响应时间关注很高，而下载网站则更加关注吞吐量。因此，在整个测试过程中，性能指标应该尽量考虑全面，这个涉及测试设计人员的个人经验。

习题 4.2

1. 负载测试主要关注软件的哪些问题？
2. 负载测试主要采用哪些方法，关注哪些指标？
3. 压力测试与负载测试有什么区别？

4.3　容量测试（Volume Testing）

容量测试，也称大数据量测试，一般有两种情况：一种是针对某些系统存储、传输、统计和查询等业务进行大数据量的独立数据容量测试；另外一种是与压力测试、负载测试、可靠性测试等相结合的极限状态下的综合数据容量测试。容量测试的关键是大数据量的测试数据的准备。容量测试的目的是通过测试预先分析出反映软件系统特征的某项指标极限值（如最大并发用户数、数据库记录数等），检查系统在其极限状态下是否还能保持重要功能的正常运转。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个具有十几万条、甚至几百万条记录的大型数据库，检验该软件是否能正常运行并能生成正确的报表。

软件容量测试可以帮助软件开发商和用户了解软件系统的承载能力和提供服务的能力，如某个电子商务网站所能承受同时进行交易的联机用户数。知道了系统容量，如果不能满足设计要求，就要寻求新的解决方案。

习题 4.3

1. 容量测试主要关注软件哪些方面的特性？

2. 容量测试与压力测试、负载测试、可靠性测试和性能测试等具有什么样的关系？

4.4 性能测试(Performance Testing)

1. 概念

软件系统的性能一般是指用以标识、评估软件在特定环境下完成任务的若干指标，例如，执行效率、资源占用、稳定性、安全性、可扩展性和可靠性等。对于很多软件系统，如实时和嵌入式系统等，仅满足功能要求是不行的，还应满足性能要求，性能已成为软件质量最重要的衡量标准之一。性能测试就是对软件的运行性能指标进行测试，判断系统集成之后在实际的使用环境下能否稳定、可靠地运行。

性能测试一般通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。性能测试主要是测试软件产品在实际应用中的性能特征，不同的运行环境，测试结果会有所不同。性能测试包括的测试内容丰富多彩，可以将性能测试概括为 3 个方面，即应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，3 个方面有效、合理的结合，可以达到对系统性能全面的分析和瓶颈的预测。

2. 目标

性能测试的目的是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件，最后起到优化系统的目的。具体有：①测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助作出决策；②识别体系中的弱点。受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系的瓶颈或薄弱的地方；③系统调优。重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能；④检测软件中的问题。长时间的测试执行可导致程序发生由于内存泄露引起的失败，揭示程序中的隐含的问题或冲突；⑤验证稳定性(resilience)和可靠性(reliability)。在一个生产负荷下执行测试一定的时间是评估系统稳定性和可靠性是否满足要求的唯一方法。性能测试类型包括负载测试、压力测试、容量测试等，其中负载测试是一种性能测试，是指程序在超负荷环境中运行，测试是否能够承担；压力测试是一种性能测试，它在系统资源特别低的各种情况下测试软件系统运行情况；容量测试确定系统可处理同时在线的最大用户数。

3. 原理

软件性能主要包括时间性能和空间性能。时间性能主要是指软件的一个具体事务响应时间，响应时间的长短并无一个绝对统一的标准，如某电子商务网站，一个可以接受的响应时间标准可以是 2∶5∶10，即 2 秒内对用户的操作予以响应是非常好的，5 秒内对用户操作进行响应则认为可以接受，而若 10 秒内还无法响应用户则将导致用户抱怨。空间性能主要是指软件运行时消耗的系统资源，它直接决定了系统的最低配置和推荐配置，系统的最低配置和推荐配置越小，软件运行时消耗的系统资源越小。

为了收集系统的时间和空间性能指标，一般将被测试系统运行在典型环境中，执行测试

用例，将探针插入代码中，或使用外部探针或性能监视器进行度量。性能测试的范围很广，可分为常规的性能测试、压力测试、负载测试、可靠性测试和大数据量测试等。所谓常规性能测试是指软件在一般用户实际使用的普通环境，即正常的软硬件环境下运行、模拟生产运行的业务压力，不向其施加任何其他压力的性能测试。其他类型的性能测试将在后面专门章节介绍。

4. 方法

性能测试可通过手工或利用自动化测试工具来完成。手工测试十分昂贵和复杂，且效果难以保证，因此，性能测试一般需要有自动化测试工具支持，如 LoadRunner 等，当然，自动化测试工具不能代替手工测试，只能辅助完成性能测试。

性能测试是对软件需求规格说明中的性能需求逐项进行的测试，以验证其性能是否满足要求，如测试在获得定量结果时程序计算的精确性（处理精度）；测试其时间特性和实际完成功能的时间（响应时间）；测试为完成功能所处理的数据量；测试程序运行所占用的空间；测试负荷潜力；测试配置项各部分的协调性；测试软件性能和硬件性能的集成；测试系统对并发事物和并发用户访问的处理能力。

与一般的测试过程一样，性能测试也遵循以下过程（如图 4-1 所示）。

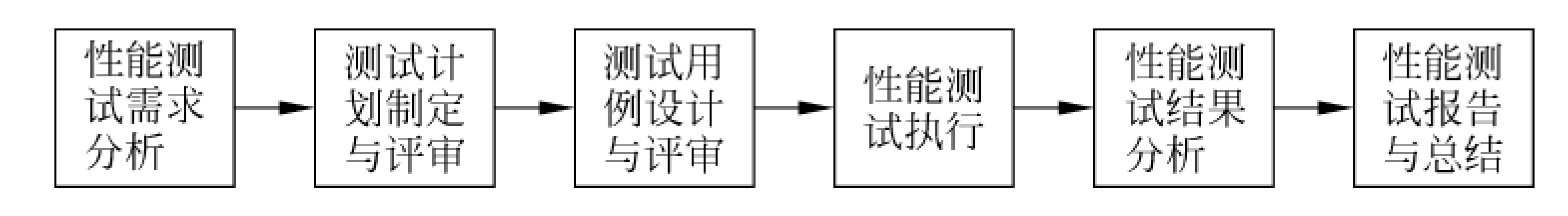


图 4-1　性能测试过程

5. 实例

针对电厂两票管理系统的性能测试，电厂工作人员可以使用该管理系统开出工作票和操作票。假设开设 100 个账户和密码可供 100 个工作人员同时开出工作票或操作票，每台机器只能由一个用户使用，每个用户只能使用各自不同的账号登录该管理系统，开票结束后，要求把工作票或操作票内容存档，若在规定的时间内没有存档，则系统强制存档。

由于一般测试部门是不可能有 100 台机器同时进行的，因此，要使用 LoadRunner 模拟 IP 地址，修改脚本来协助测试。但是，为了保证测试结果，应尽可能使用所有可用的机器进行测试，毕竟，测试工具有时是不可以完全信赖的。

6. 优缺点

性能测试是软件测试中的高端领域，要求测试人员不仅要了解系统的内部工作原理，而且要具备很强的技术能力和综合分析问题的能力。

习题 4.4

性能测试通常检查软件的哪些指标？

4.5 可靠性测试(Reliability Testing)

可靠性测试是在给被测试系统加载一定业务压力的情况下,使系统运行一段时间,以此来测试系统是否稳定。例如,可以采用 24 小时×7 天的方式连续运行系统,一般采用平均错误时间间隔来衡量被测试系统的可靠性,该值越大,系统越稳定。

1. 概念

软件可靠性是软件系统在规定的时间内以及规定的环境条件下,完成规定功能的能力。它的概率度量称为可靠度。软件可靠性是软件系统的固有特性之一,它表明了一个软件系统按照用户的要求和设计目标,执行其功能的可靠程度。软件可靠性与软件缺陷有关,也与系统的输入和系统的使用有关。理论上,可靠的软件系统应该是正确、完整、一致和健壮的。但实际上任何软件都不可能达到百分之百的正确,而且也无法精确地度量。一般情况下,只能通过对软件系统进行测试来度量其可靠性。

根据以上软件可靠性的概念:"软件可靠性是软件系统在规定的时间内及规定的环境条件下,完成规定功能的能力。"软件可靠性主要包含以下三个要素:

(1) 规定的时间。软件可靠性只是体现在其运行阶段,因此将"运行时间"作为"规定的时间"度量。"运行时间"包括软件系统运行后工作与挂起(开启但空闲)的累计时间。由于软件运行的环境与程序路径选取的随机性,软件的失效为随机事件,因此运行时间属于随机变量。

(2) 规定的环境条件。即软件的运行环境,包括各种支持要素,如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同的环境条件下软件的可靠性是不同的,有了明确规定的环境条件,才可以有效判断软件失效的责任在用户方还是开发方。

(3) 规定的功能。软件的可靠性还与规定的任务和功能有关。由于要完成的任务不同,软件的运行剖面(即软件各项功能使用情况的统计信息)会有区别,调用的子模块不同,其可靠性也就有可能不同。因此,要准确度量软件系统的可靠性必须首先明确它的任务和功能。

2. 原理

软件系统规模越做越大越复杂,其可靠性越来越难保证。应用本身对系统运行的可靠性要求越来越高,在一些关键的应用领域,如航空、航天等,其可靠性要求尤为重要,在银行等服务性行业,其软件系统的可靠性也直接关系到自身的声誉和生存发展竞争能力。特别是软件可靠性比硬件可靠性更难保证,会严重影响整个系统的可靠性。在许多项目开发过程中,对可靠性没有提出明确的要求,开发商(部门)也不在可靠性方面花更多的精力,往往只注重速度、结果的正确性和用户界面的友好性等,而忽略了可靠性。在投入使用后才发现大量可靠性问题,增加了维护困难和工作量,严重时只有束之高阁,无法投入实际使用。

(1) 软件可靠性与硬件可靠性的区别

软件可靠性与硬件可靠性之间主要存在以下区别:

① 最明显的是硬件有老化损耗现象,硬件失效是物理故障,是器件物理变化的必然结

果；软件不发生变化，没有磨损现象，有陈旧落后的问题。

② 硬件可靠性的决定因素是时间，受设计、生产、运用的所有过程影响，软件可靠性的决定因素是与输入数据有关的软件差错，是输入数据和程序内部状态的函数，更多地决定于人。

③ 硬件的纠错维护可通过修复或更换失效的系统重新恢复功能，软件只有通过重新设计。

④ 对硬件可采用预防性维护技术预防故障，采用断开失效部件的办法诊断故障，而软件则不能采用这些技术。

⑤ 事先估计可靠性测试和可靠性的逐步增长等技术对软件和硬件有不同的意义。

⑥ 为提高硬件可靠性可采用冗余技术，而同一软件的冗余不能提高可靠性。

⑦ 硬件可靠性检验方法已建立，并已标准化且有一整套完整的理论，而软件可靠性验证方法仍未建立，更没有完整的理论体系。

⑧ 硬件可靠性已有成熟的产品市场，而软件产品市场还很新。

⑨ 软件错误是永恒的，一般都是可重现的，而一些瞬间的硬件错误可能会被误认为是软件错误。

总的说来，软件可靠性比硬件可靠性更难保证，即使是美国宇航局的软件系统，其可靠性仍比硬件可靠性低一个数量级。

(2) 影响软件可靠性的因素

软件可靠性是关于软件能够满足需求功能的性质，软件不能满足需求是因为软件中的错误引起了软件故障。软件中有哪些可能的错误呢？

软件错误是软件开发各阶段潜入的人为错误：

① 需求分析定义错误。如用户提出的需求不完整、用户需求的变更未及时消化、软件开发者和用户对需求的理解不同等等。

② 设计错误。如处理的结构和算法错误，缺乏对特殊情况和错误处理的考虑等。

③ 编码错误。如语法错误、变量初始化错误等。

④ 测试错误。如数据准备错误、测试用例错误等。

⑤ 文档错误。如文档不齐全、文档相关内容不一致、文档版本不一致、缺乏完整性等。

从①到⑤，错误的影响是发散的，因此要尽量把错误消除在开发前期阶段。

错误引入软件的方式可归纳为两种特性：程序代码特性和开发过程特性。程序代码一个最直观的特性是长度，另外还有算法和语句结构等，程序代码越长，结构越复杂，其可靠性越难保证。开发过程特性包括采用的工程技术和使用的工具，也包括开发者个人的业务经历水平等。

除了软件可靠性外，影响可靠性的另一个重要因素是健壮性，对非法输入的容错能力。因此提高可靠性从原理上看就是要减少错误和提高健壮性。

3. 方法

测试可靠性是指运行应用程序，以便在部署系统之前发现并移除失败。因为通过应用程序的可选路径的不同组合非常多，所以在一个复杂应用程序中不可能找到所有的潜在缺陷。但是，可以测试在正常使用情况下最有可能的方案，然后验证该应用程序是否提供预期

的服务。如果时间允许,可采用更复杂的测试以揭示更微小的缺陷。

(1) 对构成软件的各个组件进行压力测试。压力测试是指模拟巨大的工作负荷以查看应用程序在峰值使用情况下如何执行操作。对构成软件的各个组件进行压力测试,可隔离构成组件和服务、推断出它们公开的导航方法、函数方法和接口方法以及创建调用这些方法的测试前端。

在隔离的情况下,对每个组件施加远超过正常应用程序将经历的压力。例如,以尽可能快的速度使用 1~10 000 000 循环,查看是否有暴露的问题。单独测试每个 DLL(Dynamic Link Library,动态链接库)可帮助确定组件的失败总次数。

(2) 使用集中压力测试。对每个单独的组件进行压力测试后,应对带有其所有组件和支持服务的整个应用程序进行压力测试。集中压力测试主要关注与其他服务、进程以及数据结构(来自内部组件和其他外部应用程序服务)的交互。

集中测试从最基础的功能测试开始。测试人员需要知道编码路径和用户方案、了解用户试图做什么以及确定用户运用该应用程序的所有方式。

测试脚本应根据预期的用法运行应用程序。例如,如果应用程序显示 Web 页,而且 99%的客户只是搜索该站点、只有 1%的客户将真正购买,这使得提供对搜索和其他浏览功能进行压力测试的测试脚本才有意义。当然,也应对购物车进行测试,但是预期的使用暗示搜索测试应在测试中占很大比重。

在日程和预算允许的范围内,应始终尽可能延长测试时间。不是测试几天或一周,而是要延续测试达一个月、一个季度或者一年之久,并查看应用程序在较长时期内的运行情况。

(3) 使用真实环境测试。在隔离的、受保护的测试环境中可靠的软件,在真实环境的部署中可能并不可靠。虽然隔离测试在早期的可靠性测试进程中是有用的,但真实环境的测试环境才能确保并行应用程序不会彼此干扰。这种测试经常发现与其他应用程序之间的意外的导致失败的交互。

需要确保应用程序能够在真实环境中运行,即能够在具有所有预期客户事件配置文件的服务器空间中,使用最终配置条件运行。测试计划应包括在最终目标环境中或在尽可能接近目标环境的环境中运行应用程序。这一点通常可通过部分复制最终环境或小心地共享最终环境来完成。

(4) 使用随机破坏测试。测试可靠性的一个最简单的方法是使用随机输入。这种类型的测试通过提供虚假的不合逻辑的输入,努力使应用程序发生故障或挂起。输入可以是键盘或鼠标事件、程序消息流、Web 页、数据缓存或任何其他可强制进入应用程序的输入情况。应该使用随机破坏测试来测试重要的错误路径,并公开软件中的错误。这种测试通过强制失败,可以观察返回的错误处理,以便改进代码质量。

随机测试故意忽略程序行为的任何规范。如果该应用程序中断,则未通过测试;如果该应用程序不中断,则通过测试。这里的要点是随机测试可高度自动化,因为它完全不关心基础应用程序应该如何工作。

可能需要某种测试装备,以驱使混乱的、高压力的、不合逻辑的测试事件进入应用程序的接口中。Microsoft 使用名为"注射器"的工具,可以将错误注射到任何 API(Application Program Interface,应用程序接口)中,而无需访问源代码。"注射器"可用于模拟资源失败、修改调用参数、注射损坏的数据、检查参数验证界限、插入定时延迟以及执行许多其他功能。

习题 4.5

1. 可靠性测试通常检查软件的哪些指标?
2. 可靠性测试有哪些方法?
3. 有哪些因素影响软件可靠性?

4.6 安全性测试(Security Testing)

由于社会对个人隐私的日益关注,许多软件都有特别的安全性目标。安全性测试是设计测试用例来突破程序安全检查的过程,例如,设计测试用例规避内存保护机制,破坏数据库系统的数据安全,聊天工具、电子商务等系统的安全性问题。

软件的安全性是指软件能够使得伤害或损害的风险限制在可接受的水平内,安全性是软件的内在属性,随着软件变得越来越复杂,软件的安全性也变得越来越重要,例如,基于Web的应用程序一般比常用的软件所需要的安全性测试级别要高,对于金融、电信这类特殊行业的软件而言,安全性也是最重要的软件系统性能要求之一。软件安全性做得不好,将给用户带来巨大的经济损失。

软件安全性可以分为两个级别:一是应用程序级别,包括对数据或业务功能的访问,应用程序级别的安全性可以保证在预期的安全条件下,用户只能使用应用程序的特定功能,或仅能访问有限的数据;二是系统级别,包括对系统的登录或远程访问,系统级别的安全性可以保证只有具备系统访问权限的用户才能访问应用程序。

1. 概念

安全性测试用于检测系统对非法侵入的防范能力,系统的安全必须能够经受来自多个方面的攻击。在安全性测试过程中,测试人员充当非法入侵者的角色,采用各种方法试图突破系统的安全防线,例如,尝试通过外部手段截获或破译口令;使用甚至专门开发能够瓦解防守的客户软件来攻击系统,试图破坏系统的保护机制;故意引发系统错误,导致系统失败,企图趁系统恢复的时候侵入系统。理论上讲,只要有足够的时间和资源,任何系统都可以侵入。因此,系统安全设计的基本原则是将系统设计为要攻破系统,必须付出比获得的利益更多的代价,使得非法侵入没有意义。

2. 目标

安全性(Security)测试是指在测试软件系统中危险防止和危险处理设施进行的测试,以验证其是否有效。全面检验软件在软件需求规格说明中规定的防止危险状态措施的有效性和在每一个危险状态下的反应;对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案,进行针对性测试;在异常条件下测试软件,以表明不会因可能的单个或多个输入错误而导致不安全状态。用错误的安全性关键操作进行测试,以验证系统对这些操作错误的反应;对安全性关键的软件单元和软件部件,要单独进行加强的测试,以确认其满足安全性需求。

3. 原理

在管理和维护数据库的过程中，为了保障数据库安全，我们从以下几方面限制数据库的访问安全：

(1) 限制能访问 Oracle 数据库的客户端，指定的 IP 才可以访问，防止恶意的用户登录。

(2) 即使有访问 Oracle 数据库的机会，账户的密码使用强口令和其他登录策略，恶意用户也无法轻松进入。

(3) 为每个登录账户设置了合适的权限，执行改变数据库状态的权限需要得到管理员的授权，确保了系统合法账户对数据库的操作安全，增强重要数据的抗非法访问能力。

网络服务器安全测试包括检查基础服务组件安装情况(根据系统情况合理的安装，减少安装不必要的服务控件)；查看日志是否启用，日志存储路径以及日志记录选项；检查主目录路径和目录访问权限的设置(注意：①目录建议不要和系统盘符设置在同一路径下；②目录访问权限根据所在项目系统的实际情况来设置，通常只启用"读取"权限，记录访问和索引资料权限跟系统的安全无关都默认启用，因为所用的 Internet 用户访问的目录就是 IIS 设定主目录)；默认文档的启用；访问控制的身份验证；连接超时功能的设置(可以根据项目的安全要求、参考系统需求规格说明书来进行合理的设置)；安全补丁的更新和安装情况。

网络环境安全测试主要检测的是系统所在的网络环境安全设置，此测试可以根据待测试软件本身的情况进行，或者可以忽略，主要包括以下方面：备份和升级情况；访问控制情况；网络服务情况；路由协议情况；日志审核情况；网络攻击防护情况；登录标志；安全管理等。

4. 方法

安全性测试的方法主要包括功能验证、漏洞扫描和模拟攻击等几种方法，其中功能验证是采用软件测试当中的黑盒测试方法，对涉及安全的软件功能，如用户管理模块、权限管理、加密系统、认证系统等进行测试，主要验证上述功能是否有效。

安全漏洞扫描主要是借助于特定的漏洞扫描器完成的。通过使用漏洞扫描器，系统管理员能够发现系统存在的安全漏洞，从而在系统安全中及时修补漏洞的措施。一般漏洞扫描分为两种类型：主机漏洞扫描器是指在系统本地运行检测系统漏洞的程序。网络漏洞扫描器是指基于网络远程检测目标网络和主机系统漏洞的程序。

对于安全测试来说，模拟攻击测试是一组特殊的极端的测试方法，即通过硬件或软件的方式模拟出软件可能出现的各种危险情况或故障情况，以此来验证软件系统的安全防护能力。软件安全性测试首先要考察软件安全性设计的全面性，即软件是否真的能够发现所有已知的各种危险状态并进行有效处理。

具体来说，安全性测试要考虑如下方面的内容：

(1) 资源。即软件功能和数据，应列出所有应被保护的功能和数据，确定各种功能及数据对于合法用户和潜在的侵入者具有怎样不同的价值，分析有哪些非法方法可以利用这些功能和数据。

（2）风险。即可能导致损失或伤害的事件，安全性测试中应列出所有可能的风险，将意外风险、自然风险与人为风险区别开来。一方面应分析所有风险发生的概率，另一方面应了解风险变成现实将导致的严重后果，并密切关注导致最严重后果的风险。

（3）安全性控制。针对系统中可能存在的各种风险的保护措施，其中重点关注人为风险和偶然的系统误操作，检查软件是否具有发现不安全因素的机制，并具有合理处理不安全因素的能力。例如，系统能否检测到无效参数或指令并予以适当的处理；系统能否正确保存系统配置参数，在系统发生故障时能否恢复，或能否将配置数据导出，并在其他机器上进行备份，或系统能否导入配置数据，并正常使用导入的数据；系统能否不输入密码就能登录，或对多次无效密码的输入，系统能否进行适当的处理。系统安全性测试考虑的典型问题有：执行严格的安全性功能是否比系统其他部分具有更高的有效性；系统在防止主要错误或自然意外方面的能力如何，如意外掉电时，对紧急操作的支持，随后的备份操作以及恢复到正常操作的能力；安全性控制的精度，包括错误的数量、频率和严重性等；系统对各种指令和操作的反应时间；系统对用户和服务请求处理数量的峰值和均值。

5. 实例

Web 服务作为一种分布式计算模型，以服务的形式封装应用并对外发布。由于 Web 服务往往包含了企业关键业务，若其安全性出现问题，可能会造成重大损失与严重后果。Web 服务的安全性问题成为制约其广泛应用的主要障碍。Web 服务安全性测试是保证 Web 服务安全性的重要手段。

Web 服务安全性测试框架用来指导 Web 服务安全性测试过程，可以减少测试活动的盲目性，提高测试活动的规范性，增强测试效能。Web 服务安全性测试框架包括 5 个阶段，分别是威胁建模、测试需求定义、测试策划、测试执行和报告。各阶段产生的文档主要有威胁剖面（Profile，一般理解为解剖信息）、测试需求说明书、测试策略、测试计划、测试结果与测试报告。威胁建模的主要步骤是确定安全目标、创建应用程序总体体系结构、分解应用程序、确定威胁、确定漏洞、评定威胁与漏洞等级、威胁文档化。测试需求定义是根据软件的安全需求说明与威胁剖面文档来确定安全测试的对象、内容及测试资源的分配。测试策划主要确定测试策略与测试计划。测试策略文档描述应用程序总体架构、测试目标、拟采用的测试方法、资源需求（Web Service Description Language，WSDL 文档、源代码等）、缺陷跟踪与变更控制策略。测试计划描述测试环境要求、针对威胁与安全需求设计的测试用例、人员职责分配、进度安排、测试完成标准等。最后两个阶段记录测试结果并创建测试报告。

界面因素是安全性测试技术的关键环节。在 Web 测试条件下的页面所要考虑的因素主要有容错性是否存在和正确、基本功能是否能得以实现、清单是否把所有元素列出等。例如对于动态页面同样要进行浏览走查。界面测试遵循的原则为用户界面洁净程度，在使用过程中不为用户制造障碍，布局合理，无多余功能；同一任务有多种选择，具备一定的容错处理能力等。

功能因素是安全性测试技术的重点环节。查看系统中是否有链接以及链接的位置；表单验证能否完成功能，需验证服务器正确保存数据时；后台运行的那些程序能否正确解释和使用这些信息；数据正确性验证是否达到易用性目标。对 Cookies 验证的检测，在 Cookies 中保留下的注册信息，检测需要认真核对确认 Cookies 能够正常工作并对这些资源

进行加密。而功能易用性测试在完成测试后,可以更好地了解软件的应用性,特别是听取客户的反馈意见,对程序进行改良。这里,一是使用白盒测试技术,进入代码内部,根据开发人员对代码和程序的熟悉程度,对有需要的部分进行测试;二是使用黑盒测试技术,这是根据软件需求、设计文档,模拟客户场景随系统进行实际的测试,其覆盖的全部功能,可以结合兼容性测试和性能测试等方面。

压力因素主要是考虑测试系统在基本功能通过后进行对内容的测试,进行压力测试是指实际破坏一个 Web 应用系统,测试系统的反映。压力测试是测试系统的限制和故障恢复能力,也就是测试 Web 应用系统会不会崩溃,在什么情况下会崩溃。如在对 Web 服务器进行压力测试时,可以帮忙找到死机、崩损、内存泄露等大型问题,因为那些存在内存泄漏问题的程序,通过成千上万次程序的运行,导致内存泄漏的越多,从而更进一步导致系统崩溃。另外,对压力测试工具的延伸使用可以进行系统稳定性测试、系统极限测试。

回归因素测试就是在经过一段时间后,再对以前修复过的错误重新进行测试,看该错误是否会重新出现的测试。

安全功能测试的出发点在于 Web 服务软件的功能要求是否完全实现,包括登录验证、超时限制、日志文件、目录、数据加密等,需要从应用级、传输级、消息级等展开一个完整的 Web 安全体系测试。

Web 服务常常含有一类不太引人注意的脆弱性,就是返回的错误消息往往提供过于详细的信息,而这有助于攻击者发现应用程序结构等敏感信息,例如,后端数据库的类型、采用的解析器等。因此,执行这类测试就是强制 Web 服务返回错误消息,观察错误消息的内容,分析是否存在不恰当的错误处理。

习题 4.6

结合个人实践,收集相关资料,给出一套系统的 Web 安全性测试方案。

4.7 安装测试(Installation Testing)

为了让软件具有高集成度,软件开发人员会将软件运行所需的文件打包制作成一个可执行文件,通常是.exe 的安装程序。为了让用户能正确安装软件的相应功能和组件,进而可以使用软件的功能,安装程序在制作的过程中以及发布前都需要进行测试,因此对于需要安装才能使用的软件而言,对安装程序进行安装测试是必不可少的。

有些类型的软件系统安装过程非常复杂,测试安装过程是系统测试中的一个重要部分。对于包含在软件包中的自动安装系统而言,这一点尤其重要。安装程序如果出现故障,会影响用户对软件的成功体验,用户会丧失对该软件的信心。

1. 概念

安装测试是确保软件在正常情况和异常情况下能成功安装。例如,进行首次安装、升级、完整的或自定义的安装都能进行安装。异常情况包括磁盘空间不足、缺少目录创建权限等。核实软件在安装后可立即正常运行。安装测试包括测试安装代码以及安装手册。安装

手册提供如何进行安装，安装代码提供安装一些程序能够运行的基础数据。

2. 目标

安装测试的目标是检查安装程序是否能够正确运行，软件能否正确安装，软件安装后是否可以正常运行以及完善性安装后程序是否仍然能正常运行等。安装测试需要检查软件在安装过程中写入的文件是否正确，是否漏写或者多了不应该写入的文件，同时注册表信息也必须检查，软件需要的注册表数据是否均正确写入，还包括快捷方式、卸载以及软件安装过程中易用性的测试。

3. 方法

在进行安装测试之前，测试人员需知道该软件的运行环境，以便选择合适的平台测试。此外，测试人员必须知道软件在安装过程中写入的文件与注册表信息，否则测试人员无法断定软件在安装过程中是否少写或者多写或者错写。安装测试是软件测试的一个部分，有白盒测试和黑盒测试。

1）静态白盒测试

静态白盒测试包括对软件代码等过程性描述做细致检查，包括代码审查、走查、覆盖等不执行软件的白盒测试。在安装测试中一般使用静态白盒测试，不执行安装程序，安装测试的静态白盒测试是走查安装程序的制作过程，细致检查每一个步骤中是否出错、在添加文件的时候是否多添加了或者少添加了、注册表数据添加是否齐全等。

静态白盒测试的目标是关注安装程序的整个制作过程，包括文件的添加、细化、快捷方式的创建以及注册表数据的设置。以此为基础发现安装程序在制作过程中的错误，在黑盒测试之前先做静态白盒测试可以提高安装程序的质量。

2）动态白盒测试

安装测试的动态白盒测试即是运行安装程序，核对程序安装过程是否合乎安装程序制作人员的设计。核对程序运行所需要的文件是否全部写入正确，检查注册表数据是否正确写入，快捷方式是否创建等。磁盘文件的写入可以手动找到文件写入路径，检查磁盘文件的写入，运行 Regedit 手动核对注册表数据的写入。但手动核对显然很麻烦，通常使用软件安装监视工具核对软件安装过程中文件的写入，Total Uninstall（目前最新版本为 6.0.0 下载地址：http://www.onlinedown.net/soft/21605.htm）就是一款软件安装过程监视工具。该软件能帮助人们监视软件安装的所有过程，记录下它对系统所做的任何改变，比如添加的文件、对注册表和系统文件的修改，并制作成安装前和安装后的快照。卸载软件时，不需要使用卸载程序，直接通过 Total Uninstall 便可将其完全地清除出系统，不留下任何痕迹。从而保证了系统的清洁。该软件运行截图如图 4-2 所示。

图 4-2 的左边为计算机中已经安装的程序信息，右边为某个需要查看的具体软件安装后在系统中写入的文件信息，包括文件系统、注册表、安装的服务和设备信息。测试人员可以根据右侧的详细信息很方便地知道该安装程序在安装过程中写入的信息。

3）安装程序的功能测试（黑盒测试）

安装程序的功能测试是测试被制作的安装程序的功能，指的是这个安装程序运行后是

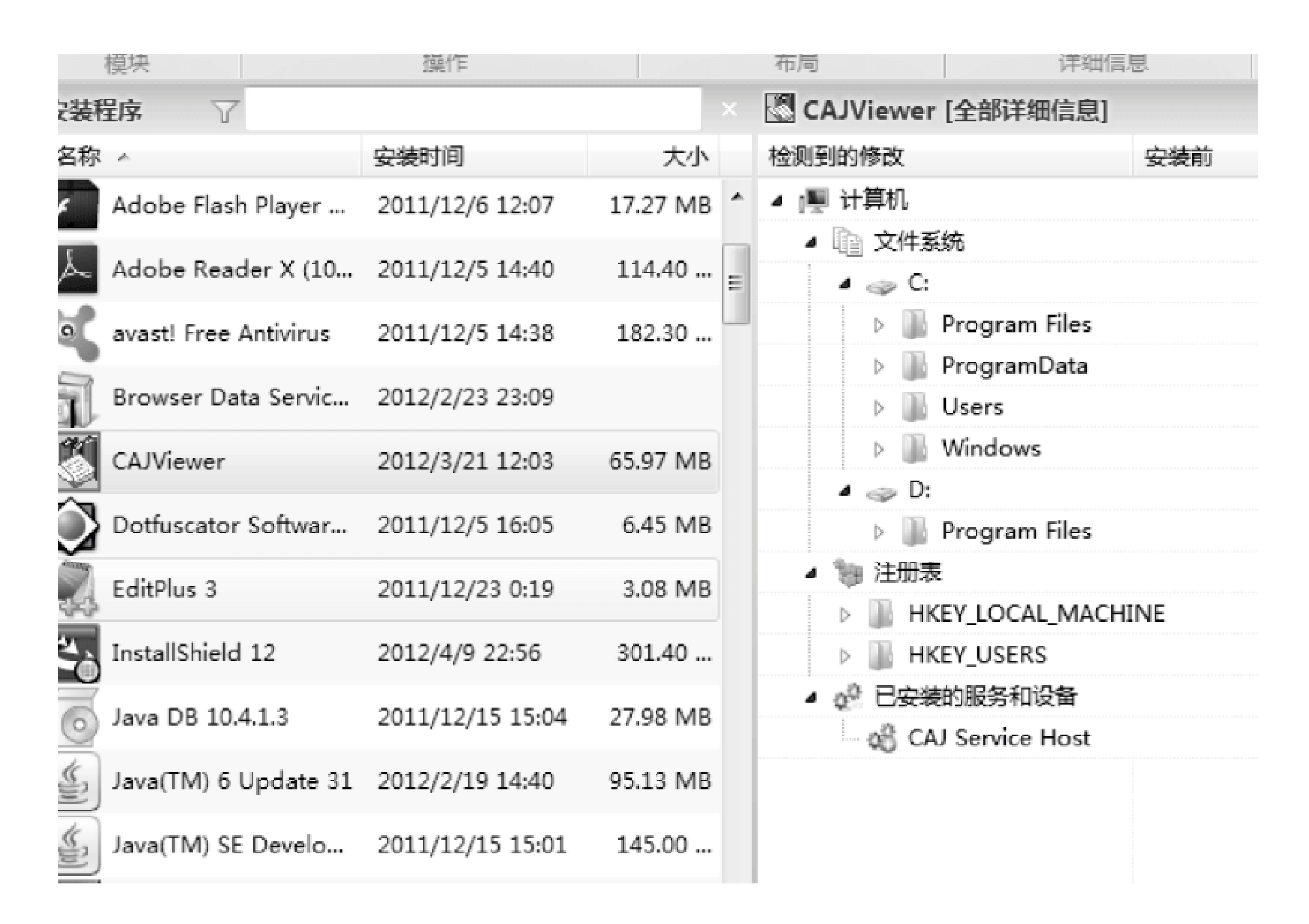


图 4-2　Total Uninstall 运行截图

否将该软件需要的文件与组件正确安装，与软件本身的功能无关。

（1）基本功能测试。对一个安装程序进行安装测试的时候，需要测试如下 9 个方面的功能，即程序信息、安装需求、安装结构、程序文件、程序快捷方式、程序注册、安装对话交互、安装本地化、安装编译。

① 程序信息。安装程序应指定程序名、公司名。程序的名字应该为有意义的单词或者首字母，能让人一下子看明白其中的意思。如果名字是一串乱七八糟的英文字母，将不便于安装人员识别。当然，也有很多的流氓软件的名字就是一串乱七八糟的字母，它们是故意这样做而起到迷惑人的目的，让人一不小心的点一下而不给任何提示的就给装了上去。

② 安装需求。如果该软件在某个操作系统下不能被使用，则在该操作系统下不应被安装，此时安装程序应该有相应的提示并能正确地退出安装。

如果该程序在运行时需要目标系统上一些特别的软件环境，则应该在安装软件时提示用户是否需要进行相应的环境安装。

如果安装路径所在磁盘空间不足，则程序应给以相应提示并请用户处理。

③ 安装结构。但凡功能较多的软件在安装的时候都应该有安装体系结构供用户选择，用户可以选择完全安装组件或者自定义安装其中的某一些组件，因为有的功能可能用户并不需要，安装了反而浪费磁盘空间以及带来更多的系统垃圾。比如 Office，我们常用的也就是 Word、Excel、PPT，而另外的那一大串，对于多数人来说都是不会用到的。

④ 程序文件。文件写入是安装测试中最重要的内容之一。测试人员可以手动地找到程序安装时写入文件的位置，逐一对比该软件在运行中所需要的文件，核实是否有缺漏或者多余的文件。如果写入的文件数量较多，人工一个一个地核对效率低下，则可以通过编写代码，让计算机核对。

特别需要注意的是，如果该软件的某个部分有依赖关系，添加了某些功能部件，而这些

功能部件在电脑中的存在位置并不在安装程序时选择的路径之下，而在安装程序制作时该部件原本的路径之下，此时需要额外的核对。

⑤ 程序快捷方式。一个好的软件，应该有快捷方式方便用户使用的，包括在开始菜单和桌面创建快捷方式，甚至有的软件是需要在快速启动栏创建快捷方式的，比如浏览器。有的软件中.exe程序不止一个，这种情况下测试人员在测试安装程序的时候一定要留意快捷方式是否选择了正确的.exe程序，如果软件的快捷方式选择到了错误的.exe上面，那么这个程序在安装后对于大多数的人来说都是无法使用进而导致放弃对该软件的使用。在进行黑盒测试的时候，测试人员必须通过点击快捷方式运行程序一遍，以确保快捷方式的正确性。

开始菜单快捷方式的目录有两个：C:\Users\USER NAME\AppData\Roaming\Microsoft\Windows\Start Menu 和 C:\ProgramData\Microsoft\Windows\Start Menu，其中后者是共享的，如果快捷方式是放在该文件夹中，则表示所有登录计算机的用户都可以使用，而前者仅由创建者控制，别的用户不可访问。

而对于桌面快捷方式，如果是在 C:\Users\Public\Desktop 下，则是共用，所有用户均可以访问；而如果是在 C:\Users\User Name\Desktop 下，则是私有，只有该用户才能访问。

⑥ 程序注册。像文件一样，注册表的写入也是程序安装过程中最重要的部分之一。程序安装时是一定会在系统注册表中写入数据的，即便这个安装程序在制作的时候并未添加注册表数据，也会在安装的时候写入注册表数据，这些系统自动写入的注册表数据测试人员不需要核对，测试人员只需要核对软件在制作安装程序的过程中设计的注册表数据是否正确。

⑦ 安装对话交互。软件安装时候的交互对话框是一个很重要的部分，因为它决定了软件安装程序的易用性。关于许可协议这一条来说，就相当于一份合同，用户同意就用，不同意则不能安装。如果该软件在使用过程中用户和软件开发公司需要遵守必要的一些规则，那么这个当然不能少。

而对于用户是否能选择安装路径这一点，除了特别底层的软件外，比如驱动，用户都应该能自己选择安装位置。用户的系统盘空间是有限的，不能什么东西都往系统盘里面放。而且东西放多了会产生垃圾导致系统变慢。还有就是重装系统后通常会把系统盘先格式化掉，那么与该软件相关联的数据也可能丢失。比如，如果 QQ 装在系统盘，那么 QQ 聊天记录就会默认在系统盘，一旦重装而没备份的话聊天记录就全没有了，万一里面有条重要信息怎么办。

⑧ 安装本地化。安装程序语言的选择上，如果是销往海外，那么英文是必须的。但是在中国，还是得有中文的为好，弄个英语的不能体现多么高端，但是会影响安装程序的易用性。不是所有的人英语都好，相当多的网民是初中毕业，实用才是最好的。可以有个英文的选择，但是中文是一定要有的。

⑨ 安装编译。安装程序的编译是根据公司的发布需求而选择的，无论安装程序被编译成什么格式，测试人员都需要至少完整的安装一遍，以保证该安装程序能正确安装。

(2) 附加功能测试。软件所需服务和设备的测试，对于大多数文件，安装程序只需要从源安装介质拷贝文件到目标系统，然后在目标系统中注册这些文件。一些特别的文件种类，

需要额外的服务和设备。比如，CAJviewer 需要安装 CAJ Service Host，而中国移动手机支付安全控件需要安装一个叫做 PassGuard 的程序。如果待测试的软件是需要安装额外的服务和设备的，测试人员可以使用 Total Uninstall 查看。

(3) 不同操作系统下的尝试。软件在本身支持的操作系统下是应该正确安装，安装后能正确运行的，那么在不支持的操作系统环境下能否正确安装呢？事实上，部分 Windows XP 下的软件，在 Win 7 中安装的时候会报兼容性问题，并且安装不成功，但这不表示就不能安装。

以 VC 6.0 为例，在 Win 7 中打开 VC 6.0 的安装程序的时候，系统是会报兼容性问题的，此时可以把这个忽略，然后直接强制性地装上去，但是这装上去也并不能用。

(4) 不同安装顺序的尝试。大部分的软件，谁先被安装谁后被安装是没有区别的，但也有例外。VS 2008 和 Office 2007 便是一个例外。先安装了 Office 2007，再安装 VS 2008，没有问题。但如果反过来安装，先安装 VS 2008，再安装 Office 2007，则在安装 Office 2007 时会报错，系统说无法找到"Office.zh-cn\officeLR.cab"文件。出现这种情况的原因是安装了 VS 2008，并安装了 SP 更新，因为其中含有 Office 2007 的更新组件，所以再用一些之前的 Office 安装文件就无法安装成功。解决的方法当然是卸载掉其中包含的 Office 2007 的组件。

习题 4.7

1. 安装几种常用软件，观察可能出现的问题。
2. 安装测试主要检查哪些问题？

4.8 可用性测试(Usability Testing)

可用性测试检查待测试软件的人机界面，通常要检查的部件包括界面布局与色彩、输入输出格式、程序流程和拼写等，以发现其中的人为因素和易用性等问题。

在做可用性测试时，可以让一群有代表性的用户尝试对产品进行典型操作，同时观察员和开发人员在一旁观察、聆听、做记录。该产品可能是一个网站，一个软件，或者其他任何产品，它可能尚未成型。测试可以是早期的纸上原型测试，也可以是后期成品的测试。

可用性最早来源于人因工程(Human Factors)。人因工程又称工效学(Ergonomics)，起源于二战时期，设计人员研发新式武器时研究如何使用机器、人的能力限度和特性，从而诞生了工效学，这是一门涉及多个领域的学科，包括心理学、人体测量学、环境医学、工程学、统计学、工业设计、计算机等。

在可用性测试中需要考虑的问题如下：

(1) 每个用户界面是否都根据用户的智力、教育背景和环境要求而进行了调整。

(2) 程序的输出是否有意义，不模糊且没有杂乱信息。

(3) 错误诊断信息是否直接，易于理解。

(4) 整体用户界面是否在语法、惯例、语义、格式和风格等方面具有完整性和一致性。

(5) 在准确性极为重要的环境下，如银行系统，输入中是否有足够的冗余信息，程序是

否易于使用等。

习题 4.8

评价几个常用软件或网站的可用性。

4.9 稳定性测试(Stability Testing)

在软件测试中，稳定性测试方法主要尝试检测软件在其使用过程中性能是否稳定，以及是否会崩溃，这种方法类似于制药领域，检测一个药品在其寿命期内能否很好地保持其品质。下面以服务器稳定性测试为例。

服务器稳定性是最重要的，如果在稳定性方面不能够保证业务运行的需要，再高的性能也是无用的。

正规的服务器厂商都会对产品进行不同温度和湿度下的运行稳定性测试。重点要考虑的是冗余功能，如数据冗余、网卡冗余、电源冗余、风扇冗余等。

一些测试方法主要分以下几种：

(1) 压力测试。已知系统高峰期使用人数，验证各事务在最大并发数(通过高峰期人数换算)下事务响应时间是否能够达到客户要求；系统各性能指标在这种压力下是否还在正常数值之内；系统是否会因这样的压力导致不良反应(如宕机、应用异常中止等)。

① 压力测试增量设计。如并发用户为 75 人，系统注册用户为 1500 人，以 5%～7%作为并发用户参考值。一般以每 15 秒加载 5 人的方式进行增压设计，该数值主要参考测试加压机性能，建议运行几次。以事务通过率与错误率衡量实际加载方式。

② 压力测试增量设计目标。寻找以增量方式加压系统性能瓶颈位置，抓住出现的性能拐点时机，一般常用参考点击率与吞吐量、CPU、内存使用情况综合判断。模拟高峰期使用人数，如早晨的登录、下班后的退出、工资发送时的消息系统等。

(2) 稳定性测试。已知系统高峰期使用人数、各事务操作频率等，设计综合测试场景，测试时将每个场景按照一定人数比率一起运行，模拟用户使用数年的情况。并监控在测试中，系统各性能指标在这种压力下是否能保持正常数值；事务响应时间是否会出现波动或随测试时间增长而增加；系统是否会在测试期间内发生如宕机、应用中止等异常情况。

根据上述测试中，各事务条件下出现性能拐点的位置，以确定稳定性测试并发用户人数。仍然根据实际测试服务器(加压机、应用服务器、数据服务器三方性能)，估算最终并发用户人数。

(3) 容错性测试。通过模拟一些非正常情况(如服务器突然断电、网络时断时续、服务器硬盘空间不足等)，验证系统在发生这些情况时是否能够有自动处理机制以保障系统的正常运行或恢复运行措施。如有 HA(自动容灾系统)，还可以专门针对这些自动保护系统进行另外的测试。验证其能否有效触发保护措施。

(4) 测评测试是用于获取系统的关键性能指标点而进行的相关测试。主要是针对预先没有明确的预期测试结果，而是要通过测试获取在特定压力场景下的性能指标(如事务响应

时间、最大并发用户数等)。

① 评测事务交易时间。为获取某事务在特定压力下的响应时间而进行的测试活动。通过模拟已知客户高峰期的各压力值或预期所能承受的压力值,获取事务在这种压力下的响应时间。

② 评测事务最大并发用户数。为获取某事务在特定系统环境下所能承受的最大并发用户数而进行的测试活动。通过模拟真实环境或直接采用真实环境,评测在这种环境下事务所能承受的最大并发用户数。判定标准阈值需预先定义(如响应时间、CPU 占用率、内存占用率、已出现点击率峰值、已出现吞吐量峰值等)。

③ 评测系统最大并发用户数。为获取整个系统所能够承受的最大并发用户数而进行的测试活动。通过预先分析项目各主要模块的使用比率和频率,定义各事务在综合场景中所占的比率,以比率方式分配各事务并发用户数。模拟真实环境或直接采用真实环境,评测在这种环境下系统所能承受的最大并发用户数。

④ 评测不同数据库数据量对性能的影响。针对不同数据库数据量的测试,将测试结果进行对比,分析发现数据库中各表的数据量对事务性能的影响。得以预先判断系统长时间运行后,或某些模块客户要求数据量较大时可能存在的隐患。

习题 4.9

稳定性测试需要综合考虑很多因素,它与可靠性测试有什么差别?

4.10 本地化和国际化测试(Localization and Internationalization Testing)

很多软件不只是服务某一个国家或者只针对某一种语言发布的,而是面向全世界的。例如,微软公司的 Windows XP 就支持 100 多种语言和方言,从南非语到匈牙利语到祖鲁语等。其他的软件公司也在做同样的事情,因为它们都意识到英语的市场还不到潜在用户市场的一半,所以必须为全球范围内的用户设计和测试软件产品。

读者可能读过一些翻译的不是很好的关于电器或者关于玩具的说明书,一看就知道这个公司没有在翻译上认真下工夫,人们可能会因此对公司的能力和实力产生怀疑。良好的翻译可以让本来就很好的产品增色不少,然而,这是一件不容易的事情。例如同样是英语,有美式英语、加拿大英语和澳大利亚英语,它们对同样一件事情或物品往往有不同表达。因此,在软件设计中除了语言本身,还要考虑用户所在的国家、地理位置和文化等因素,这个过程称为软件的本地化,有时也称软件的国际化,测试软件这方面的特性的过程称为软件的本地化和国际化测试。

如果要测试一个软件的本地化工作,测试小组最好能懂这个地方的语言,当然,不懂也没有关系,可以委托当地的测试公司做这件事情。世界各地有很多这样的软件本地化测试公司可以专门做这件事情。

当某些文本翻译成另外一种文字的时候，文本的长度会发生变化，相应的窗口、文本框或按钮的尺寸就要做适当调整，否则会影响美观，特别是文本长度变长的情况下，超出的部分可能就无法显示。有些时候这些文本在系统内部的表示也不一样，程序员在编写软件时必须考虑这些文字的内部表示，为它们留出适当的空间。

(1) 英文中一些热键和简写，在其他语言中要做相应的修改。例如，英文中查找的快捷键是 Alt+S，当然，在法语中就要做相应修改。

(2) 扩展字符问题。所谓扩展字符是指键盘上没有的字符，测试这些字符的剪切、拷贝和粘贴等在程序中的使用。

(3) 要注意字符计算和排序问题；不同文化有从左向右阅读和从右向左阅读的阅读顺序问题；在代码中要排除不同语言的文本显示问题；修改一些图片中出现的文字；软件内容要与当地文化相符合；日期格式也要符合当地习惯。

(4) Windows 提供了键盘类型和支持语言种类选项，可以根据本地化测试需求选择相应的语言和键盘进行测试，同时要注意不同语言平台中的文件之间的兼容性转换或共享问题。

习题 4.10

本地化和国际化测试主要关注哪些方面的问题？

4.11　可访问性测试(Accessibility Testing)

应用程序可访问性是指为残障人士扫除使用障碍。在使用电脑时，有些用户也许无法看见或无法移动鼠标，或者会面临很多其他困难。美国公司在产品的可访问性方面有着悠久的历史；让应用程序能够被最广泛的客户群使用具有良好的商业意义。因此，像 IBM 等著名公司都承诺将支持世界级规范和标准的技术带给残障人士。

可访问性测试一般都是在外包测试中，特别是对美外包测试中可能碰到的一种测试类型，主要检测软件系统对于残疾人是否具有可访问性。如果软件系统是为美国政府机构设计的，就必须满足可访问性指南的规范和要求，否则将违反美国的联邦法律，Section 508 Web 指南就是这样一套规范，详细描述了 16 项具体的要求，目前已经有一些专门的测试工具来检查软件系统的可访问性，例如，Watchfire WebXACT、QTP 等。

在可访问性测试规范中使用可访问性检查列表，该列表主要包含以下几条关键原则：

(1) 输入方式可选择。键盘、鼠标、语音，以及用于残障人士的设备。

(2) 输出方式可选择。显示、声音、打印，以及将图像提示转换成声音的用户界面元素。

(3) 一致性和灵活性。与颜色、字体等用户设置保持一致。

习题 4.11

1. 为什么要进行可访问性测试？
2. 可访问性测试主要检查哪些方面的内容？

4.12 授权测试(Authorization Testing)

授权是指获得允许去访问那些只有获得允许才能被使用的资源,授权测试是一种测试授权过程健壮性的方法,该方法通过理解授权工作原理和过程,利用这些信息来试图规避授权机制。

在成功地通过认证后,测试人员获得某些系统资源的使用授权,测试人员将在获得该有效凭证之后,验证与定义的角色和权限设置相关的内容。在这个测试过程中,测试人员设法寻找方法去穿越授权模式,或者通过遍历寻找漏洞,或者寻找方法升级所赋予的优先权利。

习题 4.12

授权测试主要检测什么?

4.13 容错性测试(Fault Tolerance Testing)

容错性测试是检查软件在异常条件下自身是否具有防护性的措施或某种灾难性恢复的手段。当系统出错时,能否在指定时间间隔内修正错误并重新启动系统。容错性测试包括以下两个方面:

(1) 输入异常数据或进行异常操作,以检验系统的保护性。如果系统容错性好,系统能给出提示或内部消化掉,而不会导致系统出错甚至崩溃。

(2) 灾难恢复性测试。通过各种手段,让软件强制地发生故障,然后验证系统已保存的用户数据是否丢失,系统和数据是否能尽快恢复。

对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性;对于人工干预的恢复系统,还需要估测平均修复时间,确定其是否在可接受的范围内。容错性好的软件能确保系统不发生无法意料的事故。从容错性测试的概念可以看出,当软件出现故障时如何进行故障的转移和恢复有用的数据是十分重要的。

故障转移是确保测试对象在出现故障时能成功完成故障的转移,并能从导致意外数据损失或数据完整性破坏的各种硬件、软件和网络故障中恢复。数据恢复可确保对于必须持续运行的系统,一旦发生故障,备用系统将不失时机地顶替发生故障的系统,以避免丢失任何数据或事务。容错性测试是一种对抗性的测试过程,在这种测试过程中,将待测试软件或系统置于异常条件下,以产生故障,例如,输入输出故障或无效数据库指针和关键字等,然后调用恢复进程并监测和检查应用程序和系统,核实系统和数据已得到了正确的恢复。

为了确保恢复进程可以将数据库、应用程序和系统正确地恢复到预期的已知状态,容错性测试一般包括以下各种情况:

(1) 客户机断电、服务器断电。

(2) 通过网络服务器产生通信中断或控制器中断。

(3) 数据库指针或关键字无效,数据库中的数据元素无效或遭到破坏。

容错性测试还要使用为功能和业务周期测试创建如下测试场景：往软盘保存时，不插入软盘，或将软盘加写保护；不接打印机，但进行打印操作；客户机断电或服务器断电；网络通信中断，如可以断开通信线路的连接，关闭网络服务器或路由器的电源；控制器被中断、断电或以控制器的通信中断，模拟与一个或多个控制器及设备的通信，或实际取消这种通信。一旦实现了上述场景，然后采用相应的下一步骤，检查系统的恢复功能。

习题 4.13

容错性测试一般考虑哪些方面的问题？

4.14　一致性测试(Conformance Testing)

一致性测试或类型测试是测试一个产品在效率或互通性方面是否符合某个指定的标准。为此，很多标准化组织或第三方机构建立很多测试过程和测试装置，专门测试与某些标准的一致性。

为了更好保证一致性，一致性测试通常由外部组织执行，而且这些组织往往是标准制定组织。通过这种方式测试之后，软件产品可以广而告之获得符合某个标准认证的。服务供应商、设备制造商和供应商通过一致性过程，并依赖于这些数据来保证服务质量。

习题 4.14

为什么要进行一致性测试？

4.15　配置测试(Configuration Testing)

1. 概念

软件与硬件之间的交互测试称为配置测试，主要检测各种硬件环境是否能够支持软件的正常运行。配置测试是软件测试的一个必不可少的重要环节，它是一个使用各种硬件来测试软件操作的过程。这类测试是要检查计算机系统内各个设备或各种资源之间的相互连接和功能分配中的错误。由于组成计算机的各种内部组件，以及各种外部设备可能来自很多不同的生产厂家，要保证软件能够适用于尽量多样化的硬件组合，就必须对软件进行配置测试。

配置测试的工作量一般情况下非常巨大。例如，目前市场上大致有 300 多种显卡、200 多种声卡、1500 多种调制解调器、1200 多种打印机，完全测试的组合数目至少是 300×200×1500×1200，总计有上亿种组合。规模之大难以想象。如果决定进行完整全面的配置测试，检查所有可能的制造者和型号组合，就会面临巨大的工作量。如果限于排除法测试，每一种配置单独测试一种板卡只用 30 分钟，也要 1 年的时间。这只是配置测试中的一个步骤，其他工作还有：在软件发布之前要修复软件缺陷，修复之后还要进行回归测试等。因此非常需要找出一种科学有效的方法把巨大无比的配置可能性减少到尽可能小的范围。

一般情况下，配置测试首先是确定要测试哪些硬件设备，根据待测软件的具体情况来确定所需的硬件类型；其次，确定使待测软件正常工作的各类硬件设备中，哪些硬件商标、型号和驱动程序可用，哪些硬件设备正在流行或者曾经流行，市场上有哪些主流的硬件配置，

将那些相互翻版、大同小异硬件设备划分为同一个等价区；最后将明确后的硬件配置缩减为可控制范围。

2. 方法

假设对某个待测软件(SUT)进行配置测试时，经过分析和处理，最后确定需要测试 m 类硬件设备，每类设备分别有 $a_1, a_2, \cdots, a_m$ 种硬件(通过商标、型号和驱动程序等来划分的)，这里不妨设 $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_m$。配置测试的任务是检查各种硬件设备在计算机系统内的相互连接和功能分配对待测软件的影响，检查待测软件能否在这些硬件的组合上正常工作。当 $n = a_1 \times a_2 \times \cdots \times a_m$ 比较小的时候，可以做全面试验。而一般情况下，这个值非常大，做全面试验是不可能的。因此，需要设计一种试验次数较少，效果又能与全面试验相近的试验设计方法。以下将人们在工程和科学试验中常用的试验设计方法应用于配置测试的测试用例设计。

为了便于说明问题，考虑对待测软件在比较少的主流配置上进行配置测试。这里只考虑显卡、声卡、调制解调器、打印机和主板 5 种设备，并假设每种设备有 2 个主流品牌的硬件可供选择(实际进行的配置测试要远远比这复杂)，即 $a_1 = a_2 = a_3 = a_4 = a_5 = 2$，如表 4-1 所示。

(1) 多次单因素试验法

如果要把表 4-1 中的所有情况都测试到，那么将需要 $2^5 = 32$ 个测试用例来遍历所有的测试情况。如果我们认为 32 次测试的工作量太大，那么可以采用多次单因素试验方法。所谓多次单因素试验方法，就是每次只变化一个因素的值，其他因素固定为相应的典型值。在本例中，首先对每种设备指定一个典型的主流硬件(如在显卡中，我们把 A_1 作为最典型的主流硬件)。当一种硬件设备穷举它的各种情况时，其他设备都以它典型的主流硬件来代表(在表 4-1 中，A_1，B_1、C_1、D_1、E_1 都为相应硬件设备典型的主流硬件)。那么我们将得到如表 4-2 所示的测试用例集。

表 4-1　配置测试所需的硬件类型表

显卡	声卡	调制解调器	打印机	主板
A_1	B_1	C_1	D_1	E_1
A_2	B_2	C_2	D_2	E_2

表 4-2　利用多次单因素试验法产生的测试用例集

$$
\left.\begin{array}{l}
\left\{\begin{array}{ccccc} A_1 & B_1 & C_1 & D_1 & E_1 \\ A_2 & \because & \because & \because & \because \end{array}\right\} \\
\left\{\begin{array}{ccccc} A_1 & B_1 & C_1 & D_1 & E_1 \\ \because & B_2 & \because & \because & \because \end{array}\right\} \\
\left\{\begin{array}{ccccc} A_1 & B_1 & C_1 & D_1 & E_1 \\ \because & \because & C_2 & \because & \because \end{array}\right\} \\
\left\{\begin{array}{ccccc} A_1 & B_1 & C_1 & D_1 & E_1 \\ \because & \because & \because & D_2 & \because \end{array}\right\} \\
\left\{\begin{array}{ccccc} A_1 & B_1 & C_1 & D_1 & E_1 \\ \because & \because & \because & \because & E_2 \end{array}\right\}
\end{array}\right\} \text{32 个测试用例被减少到 6 个}
$$

注：其中 ¨ 号表示相应位置的值与上面一行相同。

通过这样的方法得到了6个测试用例来代替完全遍历时所需要的32个测试用例。从32个测试用例减少到6个测试用例，测试用例数下降了5倍。此时再来观察表4-2中6个测试用例的质量。这组测试用例覆盖了各个参数的所有取值；从组合数的角度出发，例子中的被测试系统有5个参数，每个参数有2个可能值，那么每两个参数间的不同组合数为 $C_5^2 \times 2^2 = 40$ 种，而表4-2中的测试用例只覆盖了30种情况。使用这种方法在这个特定例子里，我们通过大约1/5的测试用例完成了3/4的所有两两组合覆盖。这种方法实现简单，能够对主流配置进行充分的测试。但缺点是参数间的两两组合无法完全覆盖。

根据多次单因素试验方法的特点，一般情况下，对一个待测系统进行多次单因素测试时，需要的测试用例的数目是 $n=\sum_{i=1}^{m} a_i - m + 1$，因此当待测系统的因素比较多，每个因素的取值个数也比较多的时候，这种方法就很不实用。

（2）两两组合覆盖方法

两两组合覆盖方法是一种由正交试验设计方法演化而来的一种方法，这种方法更适合于软件测试。正交试验设计方法要求对待测系统中各个参数取值两两组合的等概率覆盖，而在软件测试中，只需要对各种参数组合进行覆盖就可以了，而不需要等概率。

例如，对待测系统使用两两组合覆盖方法，首先根据表4-1产生一个两两组合覆盖表，如表4-3所示，这个表就是两两组合覆盖的测试用例表。

表4-3 两两组合覆盖表

显卡	声卡	调制解调器	打印机	主板
A_1	B_1	C_1	D_2	E_1
A_1	B_1	C_2	D_1	E_2
A_1	B_2	C_1	D_1	E_1
A_2	B_1	C_2	D_1	E_2
A_2	B_2	C_1	D_2	E_2
A_2	B_2	C_2	D_2	E_1

在本例中，表4-3中的6个测试用例要比多次单因素试验方法中表4-2的6个例子的质量高，因为同样6个测试用例覆盖了所有的两两组合。

两两组合覆盖方法需要的测试用例数目 n 有个下界，即：$n \geqslant \max\limits_{1 \leqslant i \neq j \leqslant m} a_i \times a_j$，一般情况下，比正交试验设计方法所需要的测试用例数目少，在有些情况下，甚至少很多。

（3）正交试验设计方法

当设备的种类比较少，且每一类设备可供选择的硬件也比较少的情况下，可以考虑利用正交试验设计方法进行配置测试。正交试验设计方法是一种帮助设计较为合理的试验方案的科学方法，在工农业生产和科学试验中有着广泛的应用，它利用正交表进行试验设计，仅作较少次数的试验，便能得出较为明确可靠的结论。正交试验设计方法要求任意两个参数的各种取值组合等概率出现。对表4-1中的配置情况用表4-4的正交表设计可产生的相应的测试用例集。

这组测试用例实现了对所有参数的两两组合覆盖，但需要的测试用例的规模比两两组

合覆盖方法较大一些。而且一般情况下，测试用例的数量$m \geqslant \max((\max\limits_{1 \leqslant i \leqslant n} a_i)^2, n+1)$。

在软件测试中，人们利用正交试验设计法已经取得了很好的效果，但这种方法存在 3 个方面的缺点：①该方法依赖于正交表，正交表的构造还存在很多未解决的问题，在很多情况下的正交表还无法构造；②很多情况下，正交表的规模仍然很大，即需要的测试用例的规模还需要进一步减少；③软件测试中，各种因素的两两组合只要出现一次就可以了，并不需要等概率覆盖。因此在软件测试中，用得较多的是两两组合覆盖方法。

(4) 均匀设计法

均匀设计方法是我国数学家方开泰教授和王元教授在 1978 年共同提出的，特点是保证所有参数的各个取值都能取到，而且考虑到试验点在试验范围内的均匀分布。这种方法在很多领域有着广泛的应用，并取得了丰硕的成果。这里同样可以应用均匀设计方法进行软件测试的测试用例设计，用这种方法对待测试软件进行配置测试的测试用例进行设计。

均匀设计法利用均匀设计表来产生试验设计方案，并不是对于任意情形都存在相应的均匀设计表，因此需要灵活应用。对于本例就无法直接利用均匀设计表安排测试用例，而是采用拟水平法，先选用均匀设计表 $U_6^*(6^6)$，根据它的均衡性从中选出前 5 列，并将其中的 1、2、3 用 1 代入，4、5、6 用 2 代入，然后再将每一列中值 1 和 2 用相应的本例中的参数值代入，得到的表 4-5 就是一个具有很好均衡性的配置测试用例表。

表 4-4 正交表 $L_8(2^5)$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 \\ 1 & 2 & 2 & 1 & 2 \\ 2 & 1 & 1 & 2 & 2 \\ 2 & 1 & 2 & 1 & 1 \\ 2 & 2 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 1 \end{pmatrix}$$

表 4-5 利用均匀设计产生的测试用例

显卡	声卡	调制解调器	打印机	主板
$A_1(1)$	$B_1(1)$	$C_1(1)$	$D_2(2)$	$E_2(2)$
$A_1(1)$	$B_2(2)$	$C_2(2)$	$D_1(1)$	$E_1(1)$
$A_1(1)$	$B_2(2)$	$C_1(1)$	$D_2(2)$	$E_1(1)$
$A_2(2)$	$B_1(1)$	$C_2(2)$	$D_1(1)$	$E_2(2)$
$A_2(2)$	$B_1(1)$	$C_1(1)$	$D_2(2)$	$E_2(2)$
$A_2(2)$	$B_2(2)$	$C_2(2)$	$D_1(1)$	$E_1(1)$

均匀设计法可以实现对各个配置参数的值覆盖，并且当每个参数的各个值之间有序关系时，这种方法产生的试验点均匀地分布在样本空间中。当系统配置参数不是很多，但每个参数的取值个数比较多，且值与值之间有序关系时，这种方法很有效。但在配置测试中，一般情况下，不仅配置参数多，每个参数的取值个数也比较多，而且这些值之间没有什么序关系，特别是在各种配置参数的取值数不同时，就很难找到合适的均匀设计表来设计测试用例。例如，当考虑配置参数的个数多于配置参数的取值个数时，利用均匀设计法就很难找到合适的均匀设计表。

(5) 多因素组合覆盖法

对于有些比较重要的配置测试，如果做完全测试，工作量又太大。但如果只按上面介绍的一些方法来进行测试，尽管这些方法都有科学的依据，具有很好的优点，但毕竟测试的次数比较少，没有进行完全测试，一定存在很大的风险。为了把这些风险降低到最低限度，可以根据具体情况，例如，当每个配置参数的取值个数比较少时，可以选择如三三组合覆盖、四四组合覆盖等多因素组合覆盖方法进行配置测试。

例如，在表 4-1 所示的例子中就可以采用三三组合覆盖和四四组合覆盖，当选用三三组合覆盖时需要 10 条测试用例（如表 4-6 所示），用四四组合覆盖时需要 16 条测试用例（如表 4-7 所示）。这些都比完全测试时的 32 条测试用例少得多，这样一方面可以大大降低软件因不完全测试而带来的风险，另一方面软件测试的工作量又被降低到最低限度。

表 4-6　三三组合覆盖方法产生的测试用例表

No	显卡	声卡	调制解调器	打印机	主板
1	A_2	B_2	C_2	D_1	E_2
2	A_2	B_2	C_1	D_2	E_1
3	A_2	B_2	C_2	D_1	E_1
4	A_2	B_2	C_1	D_2	E_2
5	A_2	B_1	C_2	D_2	E_1
6	A_2	B_1	C_1	D_1	E_2
7	A_1	B_2	C_2	D_2	E_2
8	A_1	B_2	C_1	D_1	E_1
9	A_1	B_1	C_2	D_1	E_2
10	A_1	B_1	C_1	D_2	E_1

表 4-7　四四组合覆盖方法产生的测试用例表

No	显卡	声卡	调制解调器	打印机	主板
1	A_2	B_2	C_2	D_2	E_1
2	A_2	B_2	C_2	D_1	E_2
3	A_2	B_2	C_1	D_2	E_2
4	A_2	B_2	C_1	D_1	E_1
5	A_2	B_1	C_2	D_2	E_2
6	A_2	B_1	C_2	D_1	E_1
7	A_2	B_1	C_1	D_2	E_1
8	A_2	B_1	C_1	D_1	E_2
9	A_1	B_2	C_2	D_2	E_2
10	A_1	B_2	C_2	D_1	E_1
11	A_1	B_2	C_1	D_2	E_1
12	A_1	B_2	C_1	D_1	E_2
13	A_1	B_1	C_2	D_2	E_1
14	A_1	B_1	C_2	D_1	E_2
15	A_1	B_1	C_1	D_2	E_2
16	A_1	B_1	C_1	D_1	E_1

3. 优缺点

软件配置测试是软件测试的一个重要内容，人们在进行配置测试时遇到的最大问题就是如何从数目庞大的配置组合中选择少量有代表性配置组合。一般常用的办法是通过等价划分、选择主流配置、简化配置要求等方法将各种配置可能性降低到可控制的范围，但这些

方法往往使得有些配置无法得到测试。

试验设计方法在很多领域一直是质量控制和保证的重要标准，在软件测试中也有着很重要的应用。在网络协议测试、编译器的测试等方面组合测试方法也得到了成功的应用。由于配置测试情况比一般的软件测试更复杂，有必要根据具体的要求，结合试验设计方法的优点，进行科学的测试方案的设计（本节内容与5.1节内容结合使用）。

习题 4.15

1. 为什么要进行配置测试？
2. 常用的配置测试方法有哪些？

4.16 文档测试（Document Testing）

由于软件＝程序＋数据＋文档＋服务，文档是软件中的一个重要内容，因此文档测试是软件测试中的一个重要内容。所谓文档测试就是针对软件开发、维护的全生命周期中涉及的文档展开测试，检查其正确性、规范性、清晰性和完整性等。

文档测试的对象包括可以帮助顺利完成软件安装的安装手册、指导用户使用软件的用户手册、联机帮助、实例与模板、错误提示、授权或注册登记表和用户许可协议、软件的包装和市场宣传材料等。

对于不涉及程序运行的文档，例如授权登记表，主要是确保文档正确、完备、易于理解。而对于涉及运行程序的文档，应在运行程序时同时检查对应的文档，保证文档与程序运行结果的一致性。文档要尽量使用图形化表示，使得用户易于理解和掌握。

文档测试不同于一般的检查和审查工作，主要是针对系统提交给用户的文档进行验证，文档测试的目的是验证用户文档是正确的并且保证操作手册所描述的过程能够正确工作。文档测试需要测试人员和用户换位思考，测试人员完全站在客户的角度考虑和评价被测试系统，按照文档中的说明进行操作，进而发现问题并做好记录。测试人员主要做好以下几点：首先对整个文档进行一般的评审，然后一个一个地进行详细评审；根据所使用的文档设计多个测试用例进行验证；严格地按照文档中记述的内容使用系统；测试每一个涉及的提示和意见；测试系统中出现的所有在线帮助文档及其链接，并保证所有可能检索到的条目有相应的文档说明；客观地测试每条语句，不能想当然；保证文档覆盖所有关键用户功能；验证所有错误信息以及文档中涉及的每个样例；保证用户文档的可读性，尽量避免使用专业性过强的专业术语；针对系统中相对薄弱的区域对其进行详细说明，把系统中的缺陷并入缺陷跟踪库。

习题 4.16

文档测试主要检查什么？

4.17　兼容性测试(Compatibility Testing)

1. 概念

兼容性测试是检验被测试软件与其他软件之间能否正确交互和实现信息共享,软件的交互不仅限于在同一台计算机上运行的软件之间,也包括通过网络与远在异地的不同计算机上运行的软件进行交互。兼容性测试无法做到完全的质量保证,但对于一个项目来讲,兼容性测试是必不可少的一个步骤。

针对软件自身而言,有向前和向后兼容性问题。向前兼容是指被测试软件与其未来版本保持兼容,向后兼容则指软件与其以前版本兼容。向后兼容是对被测试软件的基本要求,否则用户以前所做的工作在新版本中打不开,这将给用户带来巨大损失。向前兼容是一个较高的要求,软件应该预留很多接口,即使很多非常流行的软件也很难做到。无论向前兼容还是向后兼容都是限定在一定范围内的兼容,不需要考虑对所有版本的兼容。

兼容性测试的另一个问题是检测被测试软件与其他应用程序的兼容性问题。在当前的操作平台上,使用的应用程序种类繁多,被测试软件能否与它们兼容?当然没有必要检测被测试软件与所有这些软件之间的兼容性,只需选择与被测试软件关系最密切的、最重要的应用程序,并选择不同版本组合成测试用例来展开测试。例如,我们测试一套网络软件系统,需要对当前市面上流行的多种网页浏览器软件以及它们的不同版本是否兼容进行测试。

数据共享是兼容性测试的一个重要内容,在应用程序之间共享数据是对用户友好的表现,所开发的软件应符合公开的标准和规范,应允许软件与其他相关应用程序之间方便地交互数据。针对数据共享的兼容性测试主要考虑以下方面的问题:文件是否能够正常保存和读取,包括从硬盘、U 盘、光盘等各种存储介质读取和存入这些介质;文件的正常导入和导出;能够支持剪切、复制和粘贴操作;支持软件不同版本间的数据转换,用户可以在新版本中使用原来的文件,反之亦可,例如,在 Office 2003 中使用 Office 2007 文件。

2. 例子

(1) 浏览器兼容性问题

目前 IE、360、SouGou、Chrome、FireFox 等众多浏览器占领着整个浏览器市场,它们都完成着网络浏览的工作却又各有特色,各放异彩的同时也带来了网站对浏览器兼容性的问题。浏览器兼容性问题又被称为网页兼容性或网站兼容性问题,指网页在各种浏览器上的显示效果可能不一致而产生浏览器和网页间的兼容问题。浏览器兼容性问题的产生原因是,不同浏览器使用内核及所支持的 HTML 等网页语言标准不同,以及用户客户端的环境不同(如分辨率不同、缩放程度不同等)造成的显示效果不能达到理想效果。

(2) 浏览器兼容性测试工具

浏览器兼容性测试的工具多种多样,大多的原理都是调用不同的浏览器客户端程序,将其页面进行更好地展示,方便用户查看和比较,有些工具进行了一些智能的判断,对其明显的兼容性问题进行了提示。

Superpreview,微软发布的网页开发调试工具,自带有很多元素查看工具,如箭头、移

动、辅助线、对比等，在查看网页的 IE 6/IE 7/IE 8 不同表现的同时，可以对比效果。在 SuperPreview 中，人们可以同时浏览网页在各个版本的 IE 中的效果，对页面排版进行直观的比较。SuperPreview 的可用 IE 版本视系统已安装 IE 浏览器的版本而定，如果系统安装了 IE 8，那 SuperPreview 浏览器测试可用版本就包括 IE 8、IE 7 和 IE 6。

IETester 是一个免费的 WebBrowser 控件，专门用于测试网页在 IE 浏览器各个版本中兼容性的工具，版本包含 IE 5.5 至 IE 9 的各个版本，完全可以满足对 IE 兼容性的测试。

BrowserShots 在不同操作系统的不同浏览器下将网页做成截图，这是一个免费的开源工具，提供给设计师一个方便的途径来测试网站在不同浏览器下的兼容性。用户提交的网址会被加入一个任务队列，一群分布式的计算机会在浏览器中打开该网站，然后开始将截图上传到中央独立服务器供用户浏览。BrowserShots 是最有名、也是最古老的浏览器兼容性测试工具。

Spoon Browser Sandbox 也是一个很重要的浏览器兼容性测试工具，在该工具中点击所需要测试的浏览器环境，安装插件就可以进行测试了。帮助测试人员测试网页在 Safari、Chrome、FireFox 和 Opera 浏览器中是否正常。这个插件利用了虚拟机的方式，达到浏览器多版本共存，效果和浏览器的一样。

习题 4.17

为什么要进行兼容性测试？

4.18 试玩测试(Playtest)

Playtest 方法是一种游戏设计开发过程中，在游戏投放市场之前，为了检查发现游戏中潜在的错误，进一步改进游戏而采取的一种测试方法，该方法就是要求请游戏的玩家来玩这个游戏，这种方法可以是开放的、封闭的或者 β 式的。开放的 Playtest 方法是指将游戏开放给任何想加入的人来玩或者是指公司在外面招人来玩这个游戏。封闭的 Playtest 方法是指内部测试过程，不对外公布的。β 式 Playtest 方法是指在游戏投放市场之前的最后阶段，为了最后发现一些问题，而采取的一种半开放、半封闭的 Playtest 方法。

在电脑游戏、棋盘游戏或者角色扮演游戏中，Playtest 方法是非常常见的测试方法，并已成为这个领域的非常重要的质量控制过程。

Playtest 方法已经在体育运动中得到应用，足球竞技联合的发起人和创建者 Jim Foster 在 1986 年就通过一个一次性的比赛测试了他的室内足球的概念，测试证明成功之后，第二年就有 4 支球队参加了室内足球联赛。

习题 4.18

Playtest 主要用于测试游戏软件，偏重于游戏规则的测试，请选择读者自己手机上安装的一款游戏，进行此类 Playtest。

4.19　可恢复性测试(Recovery Testing)

基于计算机的系统应能在限定的时间内从失效状态中恢复过来,并继续运行,类似操作系统、数据库管理系统和远程处理系统等这类软件都有可恢复性目标。恢复性测试主要检测软件系统从软件或硬件失效中(包括程序错误、数据错误和硬件错误)恢复的能力,验证系统在应用程序执行过程中中断和回到特殊点的特性。

恢复性测试的过程中,测试人员扮演破坏者的角色,通过采用多种人工干预方式来使系统失效,从而检验系统的恢复能力。使系统能够从失效中恢复,则测试的重点在于对重新初始化、数据恢复、重启等功能的正确性验证;若系统必须通过人工干预后才能从失效中恢复,则测试重点还包括评估平均恢复时间是否在规定的范围内。

恢复性测试中需要考虑的问题包括:恢复过程是否能够正确工作;系统保护和恢复过程是否为错误提供足够的反应;是否存在潜在的灾难和已确认的系统失效,导致的后果是怎样的?恢复性测试经常采用故障注入的测试方法,即通过软件或硬件的方式模拟软件系统规定的故障模式,检查软件对硬件故障的自恢复能力。例如,在软件系统运行中断电、再加电,这种方式一致被认为是最有效的硬件故障模拟手段。在可恢复性测试中要重点考察软件运行中间状态的记录、历史状态的累计与清理等机制。

习题 4.19

1. 可恢复性测试主要检查软件哪些方面的能力?
2. 简述软件可恢复性能力的重要性。

4.20　卸载测试(Uninstall Testing)

软件能安装,当然要求能卸载,拿得起就要放得下。有的软件卸载的时候会提示用户是否保留设置,这样用户以后如果需要使用,那么再安装的时候就不需要再花时间去对软件进行设置了,比如 QQ、360 都有这功能。但有的软件比较不好,用户选择完全卸载,也不保留设置,但它就是非要在磁盘和注册表里面拥有一个毫无意义的文件夹。而网易 CC 则是会在 ProgramData 下面保留一个快捷方式,无论是使用 Windows 自带的控制面板还是 Total Uninstall 卸载都会保留。还有的软件更是奇怪,像凯立德导航,先装上它,然后卸载,之后磁盘里面会保留一个叫 Navi 的文件夹,假使把这个文件夹删掉,那么再装上凯立德导航,再使用的时候它会提示找不到 Navi 这个文件夹而无法使用。

卸载测试是对软件的全部、部分或升级卸载处理过程的测试。测试人员在测试的时候,应该在待测试程序安装之前给系统一个快照,而在卸载后再与之前的快照进行对比,这样该程序是否在安装卸载的过程中产生了垃圾就一目了然了。

因此,卸载测试主要是测试软件能否卸载,卸载是否干净,对系统有无更改,在系统中的残留与后来的生成文件如何处理等。还有原来更改的系统值是否修改回去。

习题 4.20

卸载测试主要检查软件哪些方面的特性？

4.21 能力测试（Facility Testing）

能力测试是一种典型系统测试类型，它判断目标文档提及的每一项能力是否确实已经实现，这里指的能力除了包括软件功能，还包括其他方面的内容，为了避免与功能测试发生混淆而不使用“功能”一词。

能力测试的过程是逐条语句地检查目标文档，当某条语句定义了一个要做什么，就判断程序是否满足。例如，“编码风格应该一致，程序中有充分的易于理解的注释”、“用户界面应该符合规范”等等。

能力测试与功能测试是完全不一样的，功能测试侧重检查软件各项功能本身是否被正确实现，能力测试不仅要检查各项软件功能是否齐全，还要检查像程序编码规范、代码可读性等非功能属性。

能力测试可以在不使用计算机的情况下进行，有时人工对目标和用户文档进行比较就足够了。在能力测试中一般都使用问题检查单，以保证在下一次测试时，人工检查的目标是一样的。

习题 4.21

能力测试与功能测试有什么区别？

4.22 健壮性测试（Robustness Testing）

健壮性是指在异常情况下，软件还能正常运行的能力。健壮性有两层含义：一是容错能力；二是恢复能力。健壮性测试是一种非常重要的软件质量保证方法，包括容错性测试和恢复性测试两个方面的内容。

软件的容错性测试是测试软件的鲁棒性，即软件在无效输入或压力环境条件下可以正确运行的程度。容错性测试一般通过构造一些不合理的输入来引诱软件出错，例如，输入错误的数据类型、输入定义域之外的数值等，在测试客户-服务器模式的网络软件时，关闭数据库服务器或者拔掉网线等。

恢复性测试主要测试系统在出现故障时，是否能够自动恢复，或者能够忽略故障继续运行。恢复性测试重点考察系统是否能重新运行、有无重要数据丢失以及是否毁坏了其他相关的软硬件等。

为了使系统具有良好的健壮性，要求设计人员在做系统设计时必须周密细致，尤其要注意妥善地进行系统异常处理。一个好的软件系统必须经过健壮性测试之后才能交付给用户。

习题 4.22

1. 健壮性测试应该包含哪些方面？
2. 如何在软件开发阶段为软件建立良好的健壮性？

4.23　穿越测试（By-pass Testing）

测试能否非法进入系统，例如，绕过防火墙、服务器检查进入服务器等，如图 4-3 所示，可以归为安全性测试。

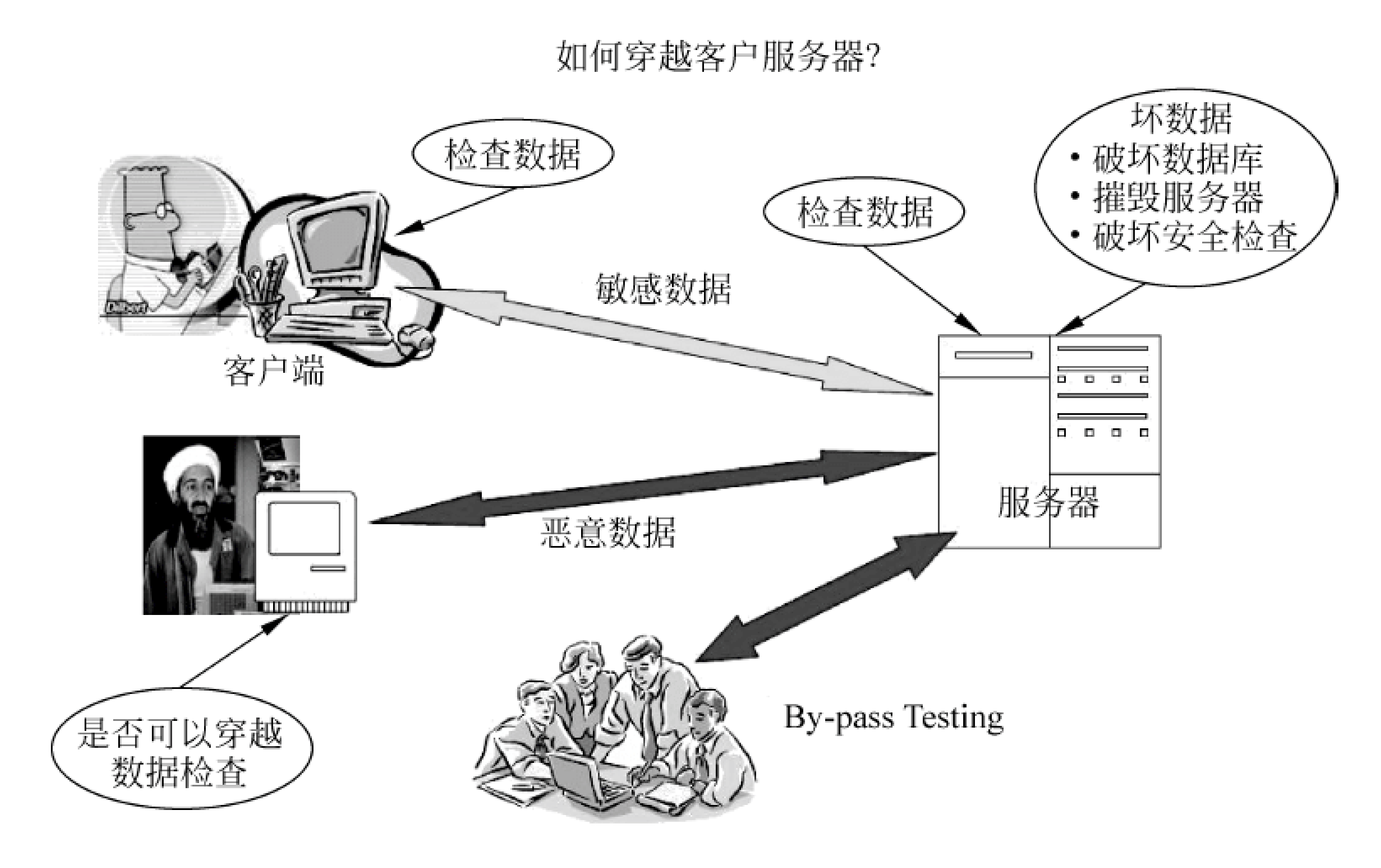


图 4-3　穿越测试的应用场景

穿越测试通过构造测试用例来故意违反各种安全约束，其目的是验证输入检查，检查系统的鲁棒性和评估系统的安全性。

穿越测试一般首先分析客户端的输入约束条件，建立各种可能的输入模型，结合典型的安全检查措施和一般的输入错误，综合利用这些信息来构造测试用例，试图通过安全检查进入服务器系统，达到破坏服务器数据库，甚至摧毁服务器系统的目的。

一个最简单也是最典型的例子就是很多人碰到的自己的电子邮件账户被盗，一般都是由于密码设置得不够安全，犯罪分子通过一些手段很快地就破解了电子邮件的账户密码。犯罪分子在破解了用户的账户密码之后，不仅获得了账户所有人很多私人秘密，有时候他们还利用这个电子邮件账户以原邮件账户所有人的名义进行诈骗。到目前为止，已经发生过无数起这种案件。

习题 4.23

检查读者自己的邮件密码、QQ 或者 MSN 的用户名和密码，计算一下破解这些密码需要多大工作量？

4.24 在线帮助测试(Online Help Testing)

用户在使用系统时候,如果出现问题,首先求助的就是在线帮助。一个糟糕的在线帮助会很大的打击用户对系统的信心。因此一个好的系统,必须要有完备的帮助体系,包括用户操作手册、实时在线帮助等。在线帮助测试(Online Help Testing)主要用于验证系统的实时在线帮助的可用性和正确性。

在线帮助给用户提供一种实时的咨询服务,一个完善的系统应该具备在线帮助的功能。在线帮助测试主要验证系统的实时在线帮助的可操作性和准确性。在进行在线帮助测试时,测试人员主要关注以下问题:

(1) 帮助文档的索引是否准确无误。

(2) 帮助文档的内容是否正确。

(3) 系统运行过程中帮助文档是否能被正常激活。

(4) 所激活的帮助内容是否与当前操作内容相关联。

(5) 是否在系统的不同位置都能激活帮助内容?帮助文档的内容是否足够详细并能解决需要解决的问题?

在实际操作过程中,在线帮助测试可以和文档测试(或资料测试)一起进行。

习题 4.24

1. 为什么需要在线帮助测试?
2. 在线帮助主要为用户提供哪些类型的帮助?

4.25 数据转换测试(Data Conversion Testing)

在实际应用中,常常会遇到环境升级的问题,同时又要保证以前的数据不能丢失,就是说要在新系统中继续使用这些数据。那么,在新系统中使用这些旧数据是否会出现问题,尤其是新系统使用了不同于老系统的数据格式时。这时一般需要进行数据转换测试,该测试的目标在于验证已存在的数据转换是否有效。在设计数据转换测试,需要考虑以下典型因素:

(1) 审计能力。需要有一个规程来进行数据转换前后的比较和分析,以保证转换的成功。保证审计能力的技术包括文件报告、比较程序和回归测试。回归测试检查验证转换过的数据不改变业务需求或引起系统出现不同的行为。

(2) 数据库验证。在把数据转换到数据库之前,需要对转换后的数据库的变化预先作评审,以确保转换方案的设计是合理的、能够满足业务需求。

(3) 数据整理。在数据转换到新系统之前,需要检查老的数据,以消除不正确的数据和矛盾的数据。

(4) 恢复计划。需要准备好回退步骤把系统恢复到以前的状态并且撤销转换操作。必须保证转换过程不会和正常的操作混杂在一起。

习题 4.25

1. 数据转换测试主要用于什么场景？
2. 数据转换测试主要检查哪些方面的内容？

4.26　备份测试（Backup Testing）

1. 概念

随着计算机产业的极速发展，各式各样的应用软件层出不穷。大量软件的出现也使得软件的质量参差不齐。软件中很重要的数据部分常常遭到人为或者意外的破坏。网络的发展也使得人们的日常生活越来越丰富。但是网络中大量流通的数据信息也相应带来了安全隐患。一旦出现数据的丢失或者损坏，小到给用户带来不好的使用体验，大到带来不可挽回的经济损失。电子设备的普及给人们的生活带来极大的方便，而其中的一些重要数据（如亲朋的联系方式、重要的备忘录等）一旦出现意外也会给人们带来很大的困扰。在这样的背景下，数据的备份便变得尤为重要。相应地，在软件投入使用前需要进行备份功能的测试。

备份测试是可恢复性测试的一个补充，是恢复性测试的一个部分。备份测试的目的是验证系统在软件或硬件失效的事件中备份其数据的能力。备份测试需要从以下几个方面来设计：

（1）备份文件，并比较新旧文件的差异。

（2）存储文件和数据。

（3）完整的系统备份过程。

（4）定时备份。

（5）备份引起的系统性能变化。

（6）手工执行备份工作的有效性。

（7）系统备份“触发器”的检测。

（8）备份期间的安全过程。

（9）备份过程期间维护处理日志的完整性。

2. 目标

在此罗列一些常见的备份功能，在备份测试的设计中可以予以参考：

（1）定时备份。定时备份是最常用的备份功能。即软件每隔一段时间或在固定的时间点自动对当前处理的数据进行备份。定时备份往往会产生多个备份文件，因此对备份文件的管理以及备份日志的维护也是定时备份测试需要着重关注的内容。

（2）手工备份。手工备份一般就是软件的“另存为”功能。在处理大数据量的软件备份测试中，尤其要关注备份内容的完整性以及备份过程对系统的影响。

（3）触发器备份。严格来说定时备份和手工备份也是触发器备份的一种，即满足某特定触发条件时系统进行的自动备份。只是这两者在备份中占的比重很高，因此单独列为一项。除此之外，还有各种触发器备份的例子，例如，数据改动一定程度、关键点更新、大数据

量计算的中间步骤等等。触发器的选择依赖于软件本身的功能，因此，为了保证测试的全面，一定要事先清楚地了解软件具有的备份功能。

(4) 针对极端/意外情况的备份。备份功能很大程度上就是为了避免意外情况的发生(例如，断网、断电、内存出错、进程崩溃等)导致的数据损坏和丢失。因此很多软件对应于应对意外情况都有额外的备份机制。对于这样的备份功能要着重考虑系统从意外中恢复之后，数据恢复的完整性。

(5) 备份内容管理。备份数据的管理以及备份日志的维护是备份功能的一个重要部分。对此有必要单独对备份功能进行测试。

3. 方法

备份测试并没有非常成体系的方法。其基本方法可分为以下 3 种：

(1) 测试员在阅读软件说明文档的基础上，了解软件具有的备份功能。选取其中一种，模拟软件需要备份的情况，观察软件是否能够成功备份数据，并对整个备份过程以及备份的结果进行评价。

(2) 测试员扮演破坏者的角色，人为的模拟各种意外情况(如断电、死机、进程出错等)，观察软件是否能成功备份数据，并对整个备份过程以及备份的结果进行评价。

(3) 多次备份之后必然带来备份文件的冗余。对于备份文件的管理和对过期备份的清理也是测试员要测试的内容之一。

4. 测试结果评估

备份不是一个独立的行为，会同时对系统的多个方面造成影响。因此，对备份测试结果的评价不能仅停留在备份本身，还要考虑多个方面的因素。评价主要考虑如下 4 个方面：

(1) 备份质量评价。备份测试最主要的测试内容，即测试软件在符合备份条件的情况下，是否能成功地备份，是否备份了所有需要备份的数据，备份的数据是否能正确地应用于原始数据的恢复。

(2) 备份过程与系统的影响。备份过程必然要占用一定的系统资源，从而对系统产生一定的影响。备份测试要观察备份过程对系统正常运作产生的影响。主要从备份过程是否影响系统的其他功能、备份是否具有足够效率两方面评价。

(3) 备份文件管理及日志。备份往往需要进行多次，对应会有多个备份的版本。为了维护版本，便于恢复时选取合适的备份文件，备份往往需要一份详尽的日志。同时，频繁的备份必定导致备份文件占据的空间越来越大。因此对备份文件的管理方式、及时清理过期的备份文件、维护良好的备份日志也是备份测试所关注的内容之一。

(4) 改善建议。结合备份测试的结果，以及在整个备份测试过程中对软件的使用体验，给出对软件备份功能的评估以及改善建议。

习题 4.26

通过对常用的本地软件(如 Word、PhotoShop 等)、网络应用(如邮箱、博客、微博、主流论坛和贴吧等)和便携的电子设备(如手机等)具有的备份功能进行测试，可以总结出比较一般的备份测试方法。并且在测试的基础上对备份的原理进行探讨，提出一定的改进意见。

4.27 接口测试(Interface Testing)

接口测试是对软件需求规格说明中的接口需求逐项进行的测试。具体测试要求包括：测试所有外部接口，检查接口信息的格式及内容；对每个外部输入输出接口必须做正常和异常情况的测试；测试硬件提供的接口是否便于使用；测试系统特性(如数据特性、错误特性、速度特性)对软件功能、性能特性的影响。对所有内部接口的功能、性能进行测试。

接口测试中经常发现的问题有：软件实现与接口协议不匹配，往往是软件接口设计文档不完善；软硬件接口设计不合理，例如，采用无限循环的方式采集外部接口数据；软件输入的接口不做有效性判断；软件接口设计的健壮性不够，考虑情况不全面；接口设计缺乏可测试性，导致测试不充分。

习题 4.27

接口测试主要检查什么?

4.28 人机交互界面测试(User Interface Testing)

人机交互界面测试对所有人机交互界面提供的操作和显示界面进行测试，以检验是否满足用户要求。具体测试要求包括：测试操作和显示界面及界面风格与软件需求说明书的要求的一致性和符合性；以非常规操作、误操作、快速操作来检验人机界面的健壮性；测试对错误命令或非法数据输入的检测能力与提示情况；测试对错误操作流程的检测与提示；对照用户手册或操作手册逐条进行操作和观察。

人机界面测试还应注意“非界面式”的人机交互部分，例如，启动按钮、操作杆和复位键等。在人机界面测试过程中，不能仅测试界面的设计是否符合规格说明，还应该加强对异常操作、误操作和非法操作等情况的测试，因为在实际使用过程中，用户可能会出现任何方式的使用。所以，人机界面测试的用例设计一定要充分考虑用户的需求，包括用户的类型、用户的使用习惯等，必要时这些用例的编写或评审应该邀请用户参与进来。而且测试中不能忽视风格、美观和易用性问题，这些要求未必都在规格说明中予以规定，但这些内容却是影响用户对软件进行评价的关键因素。

习题 4.28

人机界面测试主要检查哪些方面的内容?

4.29 余量测试(Remainder Testing)

余量测试是对软件是否达到需求规格说明中要求的余量的测试，若无特别说明，一般至少留有20%的余量。具体测试要求包括：全部存储量的余量；输入输出及通道的余量；功

能处理时间的余量等。

余量设计本身是软件安全性和可靠性设计的准则之一，目的是为了保证程序执行过程中遇到资源以外的情况时，仍能维持正常运行状态。针对余量的测试，主要进行 2 个方面的考虑：

(1) 执行时间的余量。主要考察软件在正常情况下运行，是否保持在规定执行时间 80%范围内完成。

(2) 存储空间的余量。从静态和动态 2 个方面考虑，静态余量是指源代码编译链接后的可执行代码在存储器中所占用的部分是否留有余量；动态余量是指软件运行过程中对所有相关存储器的使用不得大于规定的量。

习题 4.29

为什么要进行余量测试？

4.30 协议测试(Protocol Testing)

协议测试已经成为计算机网络和分布式系统协议工程中最重要的组成部分，由于协议标准是使用自然语言描述的，不同的生产商会有不同的理解，从而造成实现上的差异，甚至是错误的实现，所以有必要对实现的协议的正确性和有效性进行判别，这就是协议测试。协议测试包括 4 个方面的测试：

(1) 一致性测试(Conformance)。检测所实现的系统与协议规范符合程度。

(2) 性能测试(Performance)。检测协议实体或系统的性能指标(数据传输率、连接时间、执行速度、吞吐量、并发度等)。

(3) 互操作性测试(Interoperability)。检测同一协议不同实现版本之间、或同一类协议(如电子邮件协议 X. 400 和 SMTP)不同实现版本之间互通能力和互连操作能力。

(4) 坚固性测试(Robustness)。检测协议实体或系统在各种恶劣环境下运行的能力(信道被切断、通信技术掉电、注入干扰报文等)。

单纯对一个协议进行一致性测试，不能保证不同协议实现之间通信的成功，需要将协议放在一个实际的通信环境中进行测试，检测该协议与其他系统之间是否可以进行成功可靠的通信，即互操作性测试。一致性和互操作性测试都是功能测试；性能测试的目的是检查协议实现的性能参数；坚固性测试检查在各种恶劣环境下的性能参数。在这几个方面的测试中，一致性测试是基础，只有通过了一致性测试，后面的 3 个方面的测试才更有意义。

习题 4.30

协议测试主要测试哪些方面的内容？

4.31 内存泄漏测试(Memory Leak Testing)

所谓内存泄漏，简单地说就是申请了一块内存空间，使用完毕后没有释放掉，当程序运行很长时间，占用内存越多，最终用尽全部内存，导致整个系统崩溃。很多利用 C、C++、

Delphi 等高级语言开发的软件系统都存在内存泄漏问题。

内存泄漏可以分为以下 4 类：

(1) 常发性内存泄漏。发生内存泄漏的代码会被多次执行到，每次被执行的时候都会导致一块内存泄漏。

(2) 偶发性内存泄漏。发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生。常发性和偶发性是相对的。对于特定的环境，偶发性的也许就变成了常发性的。因此测试环境和测试方法对检测内存泄漏至关重要。

(3) 一次性内存泄漏。发生内存泄漏的代码只会被执行一次，或者由于算法上的缺陷，导致总会有一块仅且一块内存发生泄漏。比如，在类的构造函数中分配内存，在析构函数中却没有释放该内存，因此内存泄漏只会发生一次。

(4) 没有及时释放内存。程序在运行过程中不停地分配内存，但是直到结束的时候才释放内存。严格地说，这里并没有发生内存泄漏，因为最终程序释放了所有申请的内存。但是对于一个服务器程序，需要运行几天、几周甚至几个月，不及时释放内存也可能导致最终耗尽系统的所有内存。因此，我们也称这类内存泄漏为隐式内存泄漏。

从用户使用程序的角度来看，内存泄漏本身不会产生什么危害，作为一般的用户，根本感觉不到内存泄漏的存在。真正有危害的是内存泄漏的堆积，这会最终消耗尽系统所有的内存。从这个角度来说，一次性内存泄漏并没有什么危害，因为它不会堆积，而隐式内存泄漏危害性则非常大，因为较之于常发性和偶发性内存泄漏它更难被检测到。

对于不同的程序可以使用不同的方法和不同的工具来进行内存泄漏的检查，例如，可以使用 MemProof、Purify、BundsChecker 等工具。有些开发工具本身就带有内存问题检查机制，要确保程序员在编写程序和编译程序时打开这些功能。

习题 4.31

1. 为什么会出现内存泄漏？
2. 内存泄漏会导致什么后果？

第5章 特殊的软件测试技术

5.1 组合测试(Combinatorial Testing)

1. 概念

一般情况下,软件系统的某些输入参数、软硬件配置、内外部事件等因素及其相互作用都可能触发软件故障,针对这类软件故障比较普遍存在的情况,必须设计相应的测试用例,对各种因素及其相互间作用进行系统的覆盖性检测。但这种测试用例的设计和生成存在以下问题:一是由于影响软件的因素多,且各个因素的取值情况复杂,因此,所有可能的因素组合情况形成了一个数量巨大的测试用例空间,穷尽测试已不可能;二是手工设计和选择测试用例难度大,容易出错。

组合测试是一种充分考虑各种因素及其相互作用的科学实用的软件测试方法,这种方法克服了正交实验设计、均匀设计等传统方法的不足,设计一组数量较少的测试用例,直接检测各种影响因素及其组合对软件产生的影响,更适合于软件测试。

2. 方法与实例

作为一种复杂的逻辑体,软件系统的正常运行受到很多因素的影响,设影响待测软件(Software Under Testing,SUT)的参数(因素)有 n 个,记为 $C=\{c_1,c_2,\cdots,c_n\}$,这些参数可以是 SUT 的配置参数、内部事件、用户输入参数以及外部事件参数等。以下不妨设 SUT 的每个参数 c_i 可在有限离散点集 T_i 中取值,该集合中有 a_i 个元素,$a_i=|T_i|(1\leqslant i\leqslant n)$,不妨设 $a_1\geqslant a_2\geqslant\cdots\geqslant a_n$,并假设这些参数是相互独立的,即某个参数的具体取值不会影响其他参数的取值或存在性。用 R 表示覆盖需求,表示各个参数间可能存在的影响系统的相互关系,$R\subseteq 2^C$,例如,如果 $\{c_1,c_2\}\in R$,说明参数 c_1 与 c_2 之间存在相互作用,即参数 c_1 与 c_2 之间 $a_1\times a_2$ 个值的组合可能触发软件故障,因此在测试中要进行覆盖。在 R 中约定:如果 $\{c_1,c_2\}\in R$ 且 $\{c_1,c_2,c_3\}\in R$,在 R 中只保留 $\{c_1,c_2,c_3\}$,三个参数间存在影响系统的相互作用,包含了其中某两个参数相互作用影响系统的情况。因此 R 中任意两个元素之间互不包含。

为了检测 SUT 中各个参数及各个参数间相互作用可能对系统产生的影响,必须设计一组测试数据,实现对 R 中各个参数的取值组合情况进行充分覆盖测试。

定义1 称 n 元组 $(v_1,v_2,\cdots,v_n)(v_1\in T_1,v_2\in T_2,\cdots,v_n\in T_n)$ 为待测系统 SUT 的一个测试数据。

SUT 所有可用的测试数据组成的集合记为 Test-all,它是由 n 元组组成的集合,即

Test-all $=\{(v_1,v_2,\cdots,v_n) \mid v_1\in T_1,v_2\in T_2,\cdots,v_n\in T_n\}=\{t \mid t\in T_1\times T_2\times\cdots\times T_n\}$，显然 $|\text{Test-all}|=a_1\times a_2\times\cdots\times a_n$。

为了便于清楚地说明问题，这里采用电信系统交换机通话功能的测试作为一个实例 SUT，它有 4 个参数可能影响该系统，分别为 c_1——呼叫种类、c_2——资费方式、c_3——接入方式和 c_4——状态。对于这 4 个参数变量的每一个参数均可以有 3 种不同的状态取值，如表 5-1 所示。

表 5-1　交换机系统的测试需求

呼叫种类	资费方式	接入方式	状　态
市话	被叫方	环路	成功
长途	集中	综合业务	忙
国际	免费	专用分组	阻塞

任一个形如(市话、集中、综合业务、阻塞)的四元组是一条测试用例，如果要把表 5-1 中的所有情况都测试到，那么将需要 $3^4=81$ 个这样的测试用例来遍历所有的测试情况。一般认为，81 次测试的工作量太大，而且没有必要进行完全测试，因此需要根据该系统的测试要求、现有的测试资源等选择测试用例集。以下针对 SUT 给出各种不同类型测试用例集的定义。

定义 2　对于 SUT，设计一个测试用例集 T_c，如果 T_c 能将 R 中各个参数间的相互关系覆盖，即对 $(c_{i1},c_{i2},\cdots,c_{ik})\in R$，参数 $c_{i1},c_{i2},\cdots,c_{ik}$ 的所有 $a_{i1}\times a_{i2}\times\cdots\times a_{ik}$ 个值组合都出现在 T_c 的每个测试用例中，称 T_c 为满足关系 R 的覆盖表。当覆盖需求 R 由所有形如 $(c_{i1},c_{i2},\cdots,c_{ik})$ 元素组成，k 固定且 $|R|=C_n^k$，即 R 是所有 k 个参数的组合，此时成 T_c 是 k 维覆盖表；如果任意 k 个参数 $c_{i1},c_{i2},\cdots,c_{ik}$ 的所有 $a_{i1}\times a_{i2}\times\cdots\times a_{ik}$ 个值组合都相等次数地出现在 T_c 的测试用例中，T_c 是 k 维正交表；如果 k 不固定，且任意 k 个参数 $c_{i1},c_{i2},\cdots,c_{ik}$ 的所有 $a_{i1}\times a_{i2}\times\cdots\times a_{ik}$ 个值组合都出现在 T_c 的测试用例中时，称 T_c 是可变力度覆盖表。

由定义 2 可知，正交表是一种覆盖表，反之，覆盖表不一定是正交表；k 维覆盖表是一种特殊的可变力度的覆盖表，反之也不一定。利用 k 维正交表作为测试用例的组合测试，仍称为正交实验，利用 k 维覆盖表作为测试用例的组合测试称为 k 维组合测试，利用可变力度覆盖表作为测试用例的组合测试称为变力度组合测试。

对于表 5-1 中交换机系统，如果令 $R=\{\{c_1,c_2\}\{c_1,c_3\}\{c_1,c_4\}\{c_2,c_3\}\{c_2,c_4\}\{c_3,c_4\}\}$，$|R|=C_4^2$，表 5-2 中每一行表示交换机系统测试的一条测试用例，此时表 5-2 就是一个 2 维覆盖表。可以看出表 5-1 中的任意两个参数的 9 个值组合都出现在表 5-2 给出的测试用例中，其实表 5-2 也是 2 维正交表，因为任意两个参数的组合都只出现一次。表 5-1 也可以作为交换机系统测试的 1 维覆盖表。

参数及参数间相互作用对软件的影响在测试用例中表现为参数值和参数值之间的组合对软件系统的影响，以下我们将此定义为值模式。

定义 3　对于待测系统 SUT，取定某 $k(1\leqslant k\leqslant n)$ 个位置上的参数值后形成的 n 元组 $m=(-,\cdots,v_{l_1},-,v_{l_2},-,\cdots,v_{l_k},-,\cdots,-)$，称 m 为待测系统 SUT 的一个 k 值模式，在不引起混淆时，简称为模式，其中“—”表示相应位置上参数取值待定。当 $k=n$ 时，对应的模式

称为全模式，即为 SUT 的一个测试数据。

由定义 3 可知，软件系统由参数或参数间相互作用触发的软件故障本质上是某个值模式引发的，当待测系统 SUT 的一条测试用例 $t=(v_1,v_2,\cdots,v_n)$ 在运行中发现问题时，由于它有 $C_n^1=n$ 个 1 值模式，$C_n^2=n(n-1)/2$ 个 2 值模式，…，$C_n^n=1$ 个 n 值模式，总共有 2^n-1 中可能的触发故障的值模式，将的所有值模式记为 M_t。

令 $M_{\text{Test-all}}=\bigcup_{t\in \text{Test-all}} M_t$ 表示待测试软件系统 SUT 所有可能的值模式，M_f 表示 SUT 中存在的触发系统故障的值模式，$M_{T_c}=\bigcup_{t\in T_c} M_t$ 表示测试用例集 T_c 覆盖的所有值模式。组合测试的任务就是通过对 T_c 的精心选择，使得它击中故障模式，即 $M_f\cap M_{T_c}\neq\varnothing$（如图 5-1 所示），从而发现软件系统中隐藏的错误。

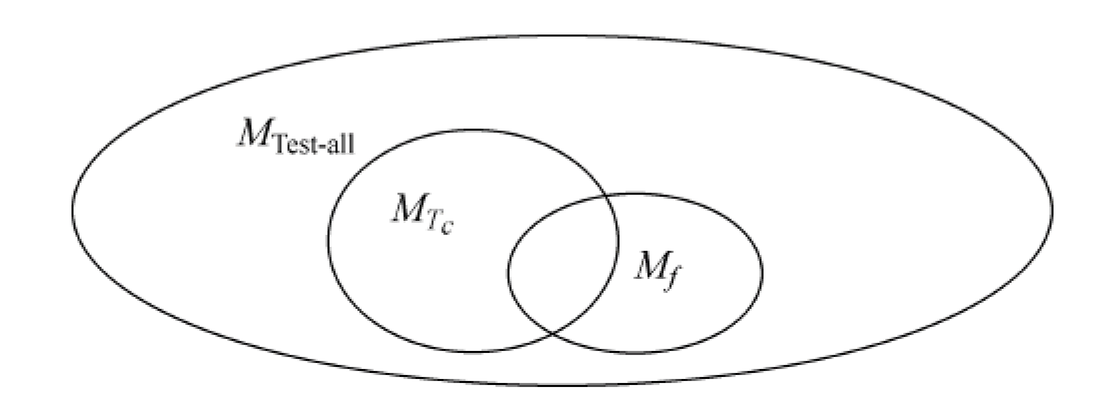


图 5-1　组合测试原理图

一开始人们利用正交表设计测试用例进行软件测试，这是将 20 世纪初期工农业生产中常用的正交试验设计方法应用到软件测试中来。随着人们对软件测试研究的深入，发现在软件测试中各种组合的覆盖率不需要相等，组合覆盖率相等的要求使得相应的正交表规模大，而且某些正交表生成存在很多困难。人们完全可以利用覆盖表来替代正交试验设计方法，随后 k 维覆盖表在软件测试中得到很好应用，k 维覆盖表生成可以发挥计算机的计算优势，成为一个新的研究热点。其实在实际软件系统中并不是任意几个参数间都存在相互作用，有些参数间根本不会有任何交互，采用 k 维覆盖表还是有点一刀切，为了降低成本，突出测试重点，人们又提出可变力度覆盖表。正交表、k 维覆盖表和可变力度覆盖表都是组合测试常用的测试用例集，组合测试实际上是试验设计方法中的一种，它们之间的关系可以用图 5-2进行描述。

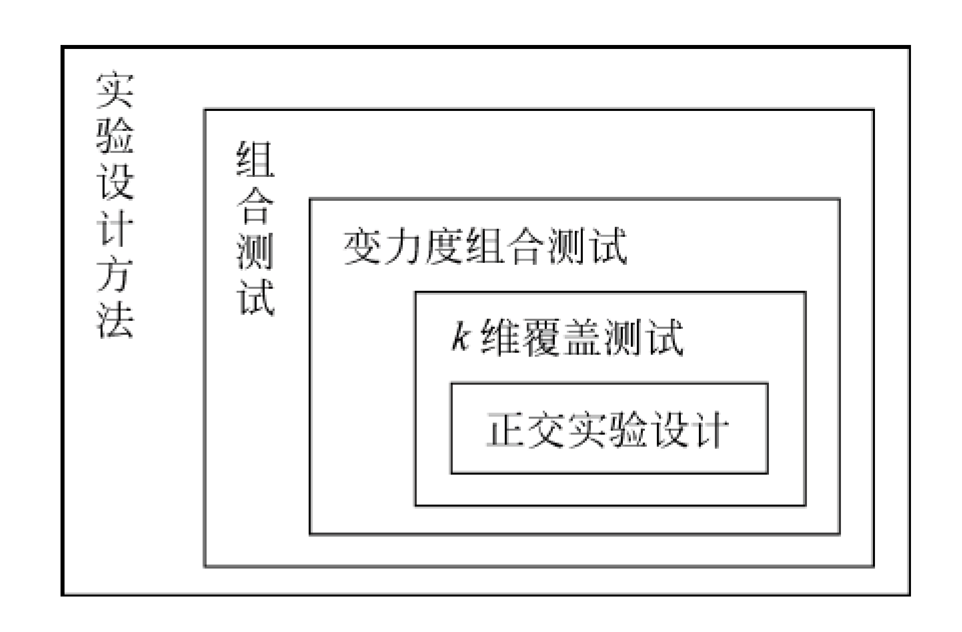


图 5-2　一种特殊的实验设计方法——组合测试的演化

在本文交换机系统的实例中，如果考虑只覆盖待测系统中的各个因素，只需要按照表 5-1 设计三个测试用例就可以了；如果考虑对任意两个因素的两两组合进行覆盖，需要选择一组两维组合覆盖的测试用例集（如表 5-2 所示），表 5-2 中每一行都给出了一个测试

用例，9条测试用例实现了对任意两个因素间所有值组合的完全覆盖，都至少出现一次。当然，要求更高时，可以设计使用三维组合覆盖测试用例（至少需要27条测试用例）、四维组合覆盖测试用例（在本例中即为完全测试）、可变力度的组合覆盖表或其他组合设计表等。

表5-2　组合测试的测试用例

呼叫种类	资费方式	接入方式	状态
市话	集中	专用分组	忙
长途	免费	环路	忙
国际	被叫方	综合业务	忙
市话	免费	综合业务	阻塞
长途	被叫方	专用分组	阻塞
国际	集中	环路	阻塞
市话	被叫方	环路	成功
长途	集中	综合业务	成功
国际	免费	专用分组	成功

3. 优缺点

组合测试的主要优点在于：

(1) 利用覆盖表作为测试用例集，覆盖表是一种旨在利用最少数量的测试用例，尽可能多覆盖各种参数值组合情况的测试用例集，从而达到以最小代价最大限度地系统检测软件系统各种因素间相互作用引发的故障。

(2) 由于很多参数间并不存在相互作用，或它们间的相互作用不会引起任何问题，相关研究显示70%左右的错误只是有一个或两个参数相互作用引起的，几乎不存在6个以上参数相互作用引发的故障，因此用覆盖表作为测试用例的组合测试有可能等效于完全测试。

(3) 组合测试很多情况下是作为一种基于规格说明的软件测试方法，只需要提取可能影响软件的参数及相应取值情况，并不需要过多的具体实现细节，因此，这是一种轻量级的软件测试方法，很容易使用。

(4) 覆盖表可以方便地自动生成，组合测试易于自动化。

像任何其他测试方法一样，组合测试方法也有很多不足，主要原因在于：

(1) 组合测试是一种不完全测试，仍然存在着很大的风险。

(2) 如果待测试软件系统的参数及其取值选择得不够恰当和准确，组合测试难以发挥作用。

(3) 如果对参数间相互作用估计不足，组合测试会遗漏测试需求。

(4) 如果没有完整的预期输出，组合测试效果也难以体现。因此，组合测试需要测试人员的正确理解、准确判断和恰当地使用。

习题 5.1

1. 组合测试的目标是什么？
2. 简述组合测试的发展过程。
3. 组合测试有哪些优缺点？

5.2 蜕变测试(Metamorphic Testing)

1. 目标

在软件测试过程中,如果测试用例的预期输出无法给出,或者难以生成,测试结果的正确与否就很难判断。蜕变测试就是用来解决这种问题,它采用待测试软件的性质,通过观察输入输出之间的关系应该满足的性质来判断测试结果的正确性。蜕变测试也可称为基于性质的软件测试,它其实是一种基于性质的软件测试。

2. 定义

蜕变测试一般指在预期输出无法给出的情况下,通过设计各种输入关系,观察输出是否满足预期的性质,用以判断软件的正确性。蜕变测试是一种基于性质的软件测试,顾名思义,就是测试软件具有的性质是否存在,通过测试软件具有的各种性质来进一步保证其质量。

3. 原理

软件测试有两个基本问题:一个是可靠的测试用例集问题,即如何生成一个充分的测试用例集,该测试用例集可以实现对软件充分有效的测试;另一个问题是测试预言问题,即如何判断测试执行结果是否正确的问题。前面很多测试方法解决第一个问题,如变异测试、组合测试、黑盒测试、白盒测试等,蜕变测试解决待测试软件测试预言无法给出或者难以给出的问题。

蜕变测试是一种测试技术,该技术根据已有未能发现软件故障的测试用例,依据一定关系或规则产生相应的一个或一组测试用例,执行这些测试用例,检查测试输出与原来测试用例的测试输出是否满足一定的性质,以确定待测试软件是否存在故障或缺陷。更具体来讲,设程序 p 在定义域 D 上实现函数 f,S 是测试人员采取的测试准则(数据流或控制流),根据这个测试准则,选取的测试用例集 $T=\{t_1, t_2, \cdots, t_n\}\subset D$, $n \geqslant 1$,运行这组测试用例产生输出:$p(t_1)$, $p(t_2)$, $\cdots$, $p(t_n)$,如果有预期输出 $f(t_1)$, $f(t_2)$, $\cdots$, $f(t_n)$,只要将它们相互比较就可以指导软件执行的对错。但是如果预期输出无法给出或者难以给出,很多情况下,就无法判断测试执行的正确与否。在这种情况下,利用输入与输出之间的一种特殊关系来进行判断,这种关系成为蜕变关系。

设程序 P 用来计算函数 f, t_1, t_2, $\cdots$, t_n 是 f 的 n 个测试输入,且 $f(t_1)$, $f(t_2)$, $\cdots$, $f(t_n)$是它们对应的预期输出结果。若 t_1, t_2, $\cdots$, t_n 之间满足关系 r 时,函数输出 $f(t_1)$, $f(t_2)$, $\cdots$, $f(t_n)$之间满足关系 rf,即:$r(t_1, t_2, \cdots, t_n)\Rightarrow rf(f(t_1), f(t_2), \cdots, f(t_n))$。显然,如果 P 是正确的,即使预期输出不知道,测试输出 $P(t_1)$, $P(t_2)$, $\cdots$,$P(t_n)$之间应该满足关系 rf,即:$r(t_1, t_2, \cdots, t_n)\Rightarrow rf(P(t_1), P(t_2), \cdots, P(t_n))$,因此,在预期输出无法获得或难以获得的情况下,通过检验输入输出之间的关系(r,rf)是否成立,来确定软件是否存在问题,这种测试方法就称之为蜕变测试。

利用蜕变测试方法测试软件时,起初给定的测试用例称之为原始测试用例,根据蜕变关系中的输入关系生成的测试用例,称为衍生测试用例。蜕变测试具有 4 个特点:为了检查程序

的运行结果是否正确，需要为程序构造蜕变关系；为了多方面验证程序正确性，就要构造多个蜕变关系；蜕变测试的原始测试用例一般根据一定的测试准则产生，衍生测试用例根据蜕变关系中的输入关系产生；蜕变测试一般与其他测试方法结合使用，如白盒测试、黑盒测试等。

蜕变测试过程描述如下：

(1) 根据测试标准，设计一组测试用例并作为源测试用例集，同时根据待测试软件设计一组蜕变关系(Metamorphic Relation) { $MR=(R,R_f)$ }。

(2) 使用源测试用例来测试被测对象，如果某一个源测试用例 t 引起了异常，则报告测试未通过。

(3) 对于每一个蜕变关系 $MR=(R,R_f)$，①从源测试用例，根据关系 R 生成新的测试用例 t'；②用 t'测试被测对象；③如果被测对象异常退出，报告测试未通过；否则④如果被测对象正常结束并输出 r_k，使用关系 R_f 验证，如果 $R_f((t_1,r_1),\cdots,(t_k,r_k))$不成立，则报告测试未通过；如果没有错误报告，则认为测试通过。

4. 实例

如果程序 P 是计算三角正弦函数值 $\sin(x)$，测试计算三角形正弦函数值 $\sin(x)$程序 P，x 可以取 36°，但其预期值 $\sin(36°)$无法给出，可以利用三角形弦函数的性质 $\sin(180°-x)=\sin(x)$，计算 $P(180°-36°)$，看看它的值是否与 $P(36°)$相等。如果不相等，很明显，该程序存在问题。往往通过一两个性质还不能保证测试的充分性，需要尽可能多地测试各种性质，例如，$\sin(90°-x)^2+\sin(x)^2=1$ 等性质。我们不知道 $\sin(29°)$的预期输出值是多少，但可以根据关系：$t_1+t_2=90°\Rightarrow\sin(t_1)^2+\sin(t_2)^2=1$，产生衍生测试输入 61。检查程序 P 在这两个输入 $P(t_1)$、$P(t_2)$上的输出是否满足以上关系 $P(t_1)^2+P(t_2)^2=1$。

5. 优缺点

尽管蜕变测试(基于性质的软件测试)可以有效地测试软件，但由于软件的各种性质获得对测试人员要求比较高，而且根据软件性质设计具体的测试用例不便于自动化。更多关于基于性质的软件测试内容请见 5.11 节。

习题 5.2

1. 蜕变测试主要解决软件测试中的什么问题?
2. 蜕变测试的主要步骤是什么?

5.3 基于规格说明的软件测试(Specification Based Software Testing)

在软件开发的每个阶段都会产生很多描述性文档，如需求分析、软件规格说明和设计等，这些文档与软件的实现应该保持高度的一致，即软件实现要与设计一致，相应地还要与需求规格说明和用户需求一致。因此，软件验证和确认技术既要保证各阶段产品的质量，又要保证整个过程的质量。基于规格说明的测试是验证软件的实现与规格说明的一致性(如

图 5-3 所示)。基于软件规格说明生成测试用例有很多好处,例如,可以尽早地在软件实现之前设计测试计划及测试用例,可以用于规格说明的推理,尽早地发现规格说明、设计和实现中可能存在的问题和不足。规格说明定义了软件系统最基本的要求和内容,它可以为测试提供很多非常有价值的信息,所以由此生成的测试用例更接近用户需求,而且可以定义测试的预期输出(如图 5-4 所示)。

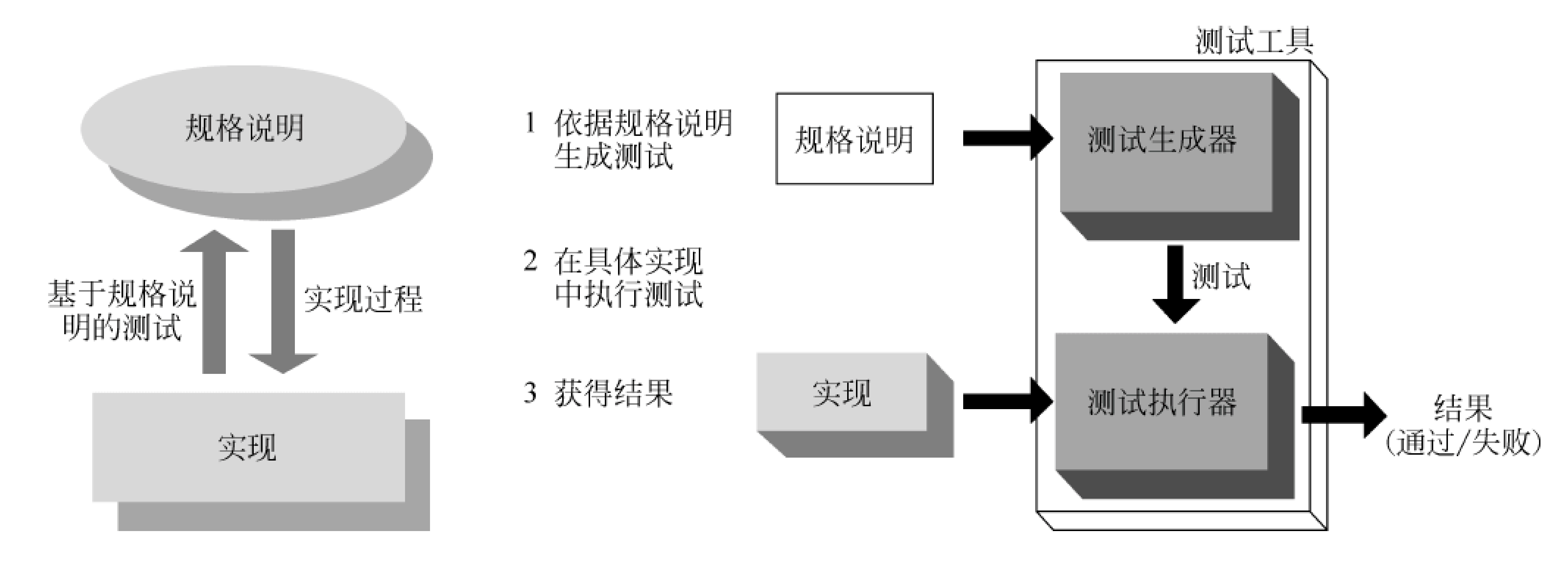


图 5-3　基于规格说明的测试　　　　图 5-4　基于规格说明的测试过程

基于规格说明的测试是一种黑盒测试技术,它直接面向于检查软件系统是否得到正确的实现,或者是否得到充分完整的实现,因此是基于实现的测试(白盒测试)的一种补充(如图 5-5 所示)。

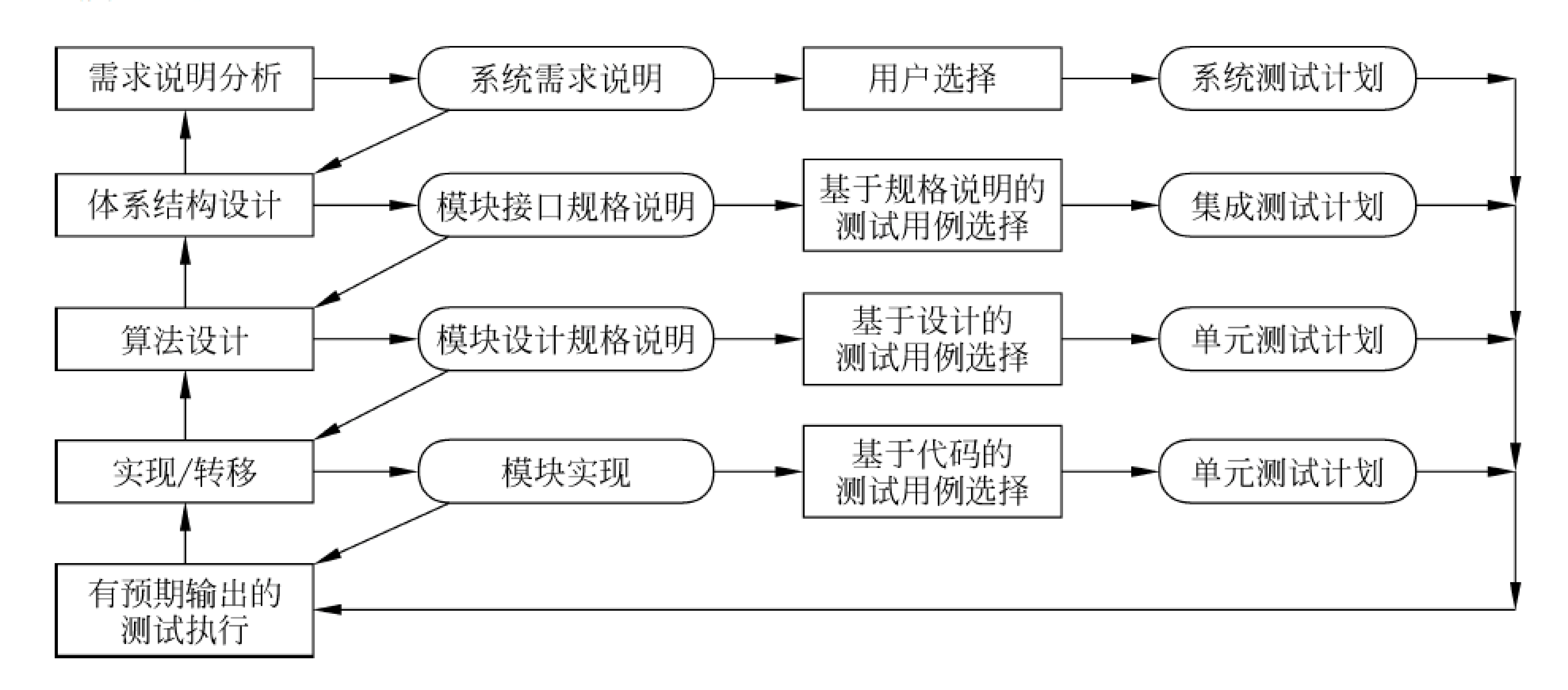


图 5-5　软件生命周期中测试用例的设计选择过程

人们关于各种形式化规格说明,如 Z 语言,提出了一些自动的测试用例生成方法,但对于非形式化的规格说明一般只能人工挑选测试用例,并产生预期输出与实际执行结果进行比对。

习题 5.3

基于规格说明的测试有哪些优点?

5.4 基于模型的软件测试(Model Based Testing)

1. 概念

在很多学科,人们都用模型来理解和描述系统,从生物学到航空领域,都用模型来理解或设计产品。在软件工程领域,模型也是人们常用的一个重要的形式化工具。

软件模型是对软件行为和软件结构的抽象描述,软件行为可以用系统输入序列、活动、条件、输出逻辑或者数据流进行描述,软件结构则使用组件图、部署图等进行描述。针对测试任务,通过对软件功能和结构进行抽象并用易于理解的方式进行描述,获得的模型就是对被测试软件系统精确的描述,可以用于软件测试。一般对软件不同行为要用不同模型进行描述,例如,控制流图、数据流图表达了程序和代码结构间的行为关系;决策表和状态机则可以描述软件外部行为。基于模型的软件测试可以根据软件行为模型和结构模型生成测试用例。当前软件规模庞大也使基于程序的测试十分困难,而基于模型的软件测试方法不仅可以有效地提高测试效率,提高测试用例生成的自动化程度,进行测试失效辨识,也有利于评价测试结果。

基于模型的软件测试方法越来越成为人们研究的一个热点,主要有 3 个原因:复杂软件的安全性等质量需求越来越高;日益流行的一些新型软件开发方式,如 UML、MDA 等,这些开发方法中使用的模型可以直接用于测试;以及测试驱动的开发越来越为人们所重视。

软件测试模型包括有限状态机、UML、文法等模型。本章 5.25 节～5.30 节分别介绍 6 种不同的基于模型的软件测试方法,而 2.2.5 节(状态转换测试)和 2.2.6 节(语法测试)也都属于基于模型的测试。

2. 方法

基于模型的软件测试方法可以分成以下几个步骤:

(1) 分析理解待测试软件。只有充分地分析理解待测试软件,才有可能构造准确的测试模型,不仅要求了解软件需求规范和设计文档、用户手册,还要与开发人员进行交流。更具体的工作包括:识别软件系统的用户、枚举每个用户的输入序列、研究每项输入的可能取值范围,包括合法值、边界值、非法值以及预期输出等。这项工作往往需要工具支持。

(2) 选择合适的测试模型。不同的模型适用于不同类型软件的测试,因此需要根据软件特点选择模型。例如,电话交换系统多使用状态模型;并发软件系统中不同组件并发运行用状态图建模;马尔可夫链可以对软件进行失效统计分析。模型的选择还依赖于软件系统的工作特点,例如,测试长期运行软件系统可以使用状态机模型。只有充分理解模型和软件系统,才能选择合适的模型对软件进行测试。以状态机为例,自动机理论可以对状态机进行分类,说明不同的状态机可以表达什么语言,从而可以根据应用程序的功能和特点,选择不同的状态机模型。由于根据有限状态机产生测试数据相当于遍历有向图,因此图论算法可以指导产生测试用例。基于模型的软件测试对测试人员的知识结构和技术水平提出了一定要求,如果一个开发组织使用模型完成需求分析和系统设计,开展基于模型的软件测试就比较容易。因为根据系统分析和设计的模型,往往可以直接应用基于模型的软件测试技术,还可以根据测试的进展不断地调整模型或模型的细节,并有利于在开发过程早期进行测试规划。另外,还可以根据开发组织使用的测试工具选择特定的模型。

(3) 构造测试模型。以基于状态机模型的测试为例说明如何构造测试模型。首先要抽

象出软件系统状态,状态抽象一般要根据输入及输出条件进行。一般包括以下过程:生成一个输入序列并说明每个输入的适用条件,称作输入约束,例如,电话未摘机时才允许有摘机动作发生。对每个输入要说明产生不同响应的上下文环境,称作响应约束,例如,电话摘机时,如果当前状态为振铃,则进行通话;否则,为拨号音。根据输入序列、输入约束和响应约束构造相应状态机模型。

(4) 生成和执行测试用例。测试用例的自动生成依赖于测试所使用的模型。以有限状态机模型为例,被遍历路径中弧的标记构成的序列就是测试用例在构造满足测试准则的路径时,必须考虑约束条件,如路径长度限制。统计测试还要考虑迁移概率,生成了满足特定的测试充分性准则的测试例集合后,就可以执行测试用例。以状态机模型为例,首先要写出仿真该软件系统的每个不同外界输入产生的脚本,称为仿真脚本,每个仿真脚本对应模型中一个不同的迁移。然后把测试用例集合翻译为测试脚本,也可以用测试生成器通过遍历状态机的迁移直接产生测试脚本,测试用例的执行就是测试脚本的执行过程。脚本是可以重复利用的资源,维护测试脚本也是测试工作的一部分。

(5) 收集测试结果进行分析。基于模型的软件测试方法并没有解决测试失效辨识(Oracle)问题,仍然要人工检查输出是否正确。但是通过状态验证可以部分解决测试失效辨识问题,状态是内部数据的抽象,比较容易验证。另外与传统测试相比,基于模型的软件测试的优势之一就是可以根据测试结果,分析软件的其他质量因素,如可靠性等。

3. 优缺点

基于模型的软件测试可以大大提高测试自动化水平,也可以部分解决测试失效辨识问题,可以进行测试结果分析,有利于测试过程的重用,并可以应用成熟的理论和技术获得比较完善的分析结果。现代软件工程强调增量和迭代的开发过程,例如,采用面向对象开发技术,提高软件开发质量和加快软件开发速度,由于面向对象软件系统的规范多数使用模型表达,这些模型中包含了大量可以用于软件测试的信息,因此,基于模型的软件测试可以把软件测试工作提前到开发过程早期,如进行测试工作的早期规划和设计。

基于模型的软件测试存在以下缺点:测试人员需要具备一定的理论基础,如状态机理论和随机过程的知识,还要掌握相关工具使用方法;需要一定的前期投入,如模型的选择、软件功能划分、模型构造等;有时无法克服模型的固有缺陷,如状态爆炸即状态空间迅速增长直至无法控制,这对检验、评审和维护模型都提出了要求,也直接影响了测试自动化。状态爆炸问题一般通过抽象和排除方法,减小测试规模和降低测试难度来解决。

习题 5.4

1. 有哪些模型可以用来做基于模型的软件测试?
2. 基于模型的软件测试具有哪些优缺点?

5.5 基于错误的软件测试(Fault Based Testing)

基于软件的系统越来越多地为人们提供一些非常关键的服务,例如,在航空、医疗和工业控制等领域的安全关键性软件、电话和网络等领域的基础设施关键性软件以及电子商务

等。由于这些软件系统的复杂性，软件系统中可能在多个方面存在一些潜在的错误，尽管软件系统一般都是按照正常和预期的行为方式构建的。基于错误的软件测试认为软件中可能存在或包含一些错误或异常，该方法目标是证实软件中是否存在事先描述的错误。

1. 概念

基于错误的软件测试是一种有效的软件测试方法，该方法针对待测试软件中可能存在的某种软件错误，设计相应的测试用例，当运行这组测试用例时，如果没有发现错误，则认为该软件中不存在这类错误。在传统的软件测试中，如果一个测试用例在运行过程中没有发现任何错误，便认为这个测试用例是无用的。但在基于错误的软件测试中，测试用例的正确执行可以帮助我们确信在软件中不存在某类错误。

变异测试可以认为是一种特殊的基于错误的软件测试方法，它按照一定机制，通过植入错误的方式，产生源程序的很多变异体，然后执行一组在源程序上已经通过的测试用例，如果这组测试用例可以发现所有变异体中植入的错误，则认为该组测试用例的测试能力是充分的，否则，要增加一些测试用例，直至发现所有植入的错误。在这个过程中，对于源程序的测试看起来只是一个副产品，变异测试主要用于评估已有测试用例的测试能力是否充分。更多关于变异测试的内容请见 5.9 节。

2. 方法

基于错误的软件测试方法首先要充分利用软件的规格说明，用来描述软件的各种形式化模型和技术，研究该软件系统的各种不同视角的视图和不同粒度的抽象，利用故障树分析等技术推测软件中可能存在的软件故障，然后，根据这些故障的特点精心设计一组测试用例，运行软件，验证软件是否存在预料中的故障。

习题 5.5

基于错误的软件测试有哪两种形式？

5.6 基于搜索的软件测试(Search Based Testing)

1. 概念

基于搜索的软件测试是一种利用元启发式搜索技术自动生成测试数据的测试方法，可以进行结构测试(如覆盖某些特定的结构元素)、功能测试(如测试某些特定的功能)、灰盒性质测试(如安全性检测)、非功能性测试(如检测最坏执行时间)。元启发式搜索技术是一个高层次框架，它利用一些启发式规则，在合理的时间成本内为复杂的组合问题求解，该框架包括爬山法、模拟退火方法、遗传算法、粒子群方法、蚁群算法等基于不同启发式规则的算法。传统的测试用例生成方法有手工生成和自动生成，都受到软件规模和复杂性等多方面因素的制约。手工生成测试用例由测试人员负责，这项工作很困难，属于劳动密集型，要耗费大量时间且容易出错。因此测试用例的自动生成非常有必要，但单纯的穷举所有测试用例是不可行的，随机的方法不可能对软件进行一些深层次的测试。当然，根本原因在于：测

试用例生成问题是一种不可判定问题。基于搜索的软件测试作为一种测试用例自动生成方法,它提供了一种有效解决方案,可以有效克服原有的自动生成方法的不足。

2. 目标

利用启发式搜索技术解决的问题一般都是 NP-complete 或者 NP-hard 问题,或者虽然这种问题理论上存在多项式时间求解方法,但实践上不实用。

3. 方法

基于搜索的软件测试不仅仅是一个算法,它也是一种针对特定问题求解的策略。对于测试用例生成,它要先将测试充分性准则转换为目标函数,利用目标函数对搜索到的可行解进行评估,并与搜索目标进行对比,从而将搜索导向搜索空间中一个有价值的区域。

4. 实例

以三角形分类程序为例(如图 5-6 所示),如果采用随机测试,一些小概率路径就很难执行到,例如,该例中等边三角形的情况就很难达到。因此在生成测试用例时,非常有必要进行导向性的搜索。

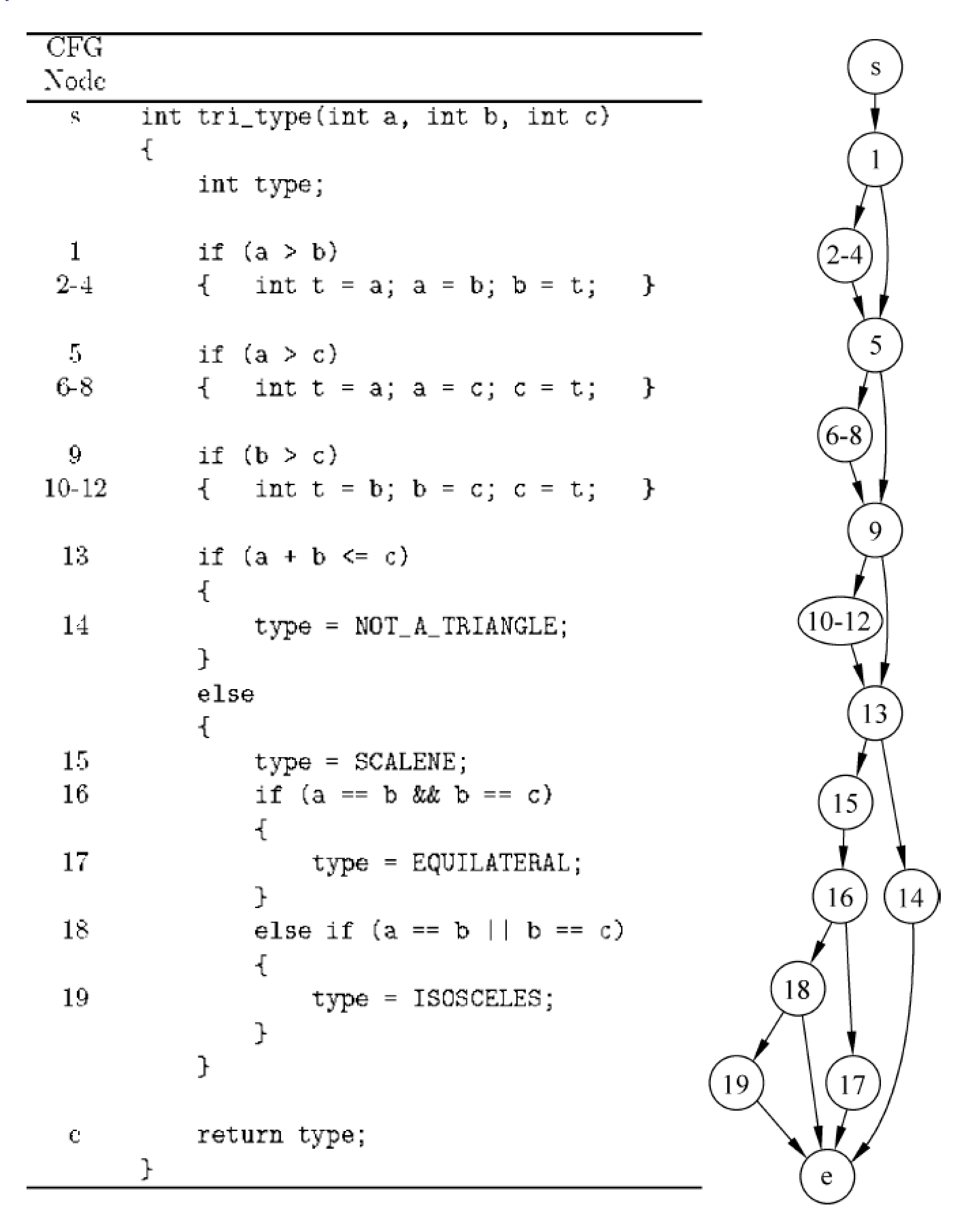


图 5-6 三角形分类程序及控制流图

Miller 和 Spooner 最早将程序的实际执行结果与一个搜索技术相结合，他们的方法已被广泛使用。该方法首先从程序中选择一条路径，形成一个只包含该条路径的程序版本，其中的分支语句用路径约束条件满足程度 $c=0$ 或 $c>0$ 或 $c\geqslant 0$ 替代，这里的 c 是对执行相应路径的条件接近程度的一个估计。以三角形分类程序为例，执行路径 $<s,1,5,9,10,11,12,13,14,e>$，对应该路径的直线程序如下：

```
Int tri-type(int a, int b, int c)
{int type;
 (c1 = (b-a))>=0;
 (c2 = (c-a))>=0;
 (c3 = (c-b))>0;
 t=b; b=c; c=t;
(c4 = (c-(a+b)))>=0;
 Type=NOT_A_TRIANGLE;
}
```

利用这些约束可以构造一个适应值函数 f，f 的值提供了所有约束条件被满足情况的一个估计：f 如果是负值，表明有一个或多个约束未被满足；如果所有约束条件被满足了，f 应该是正值。利用求最大值方法寻找输入 a、b、c，使得 f 的值越来越接近于 0，最后成为正值。

Korel 扩展了以上工作，利用程序插桩方法代替产生直线程序的方法。为了执行某条指定路径，程序一开始执行一些任意输入。在执行过程中，如果执行了一条非指定路径，该条路径与指定路径有偏差，就利用目标函数，调用局部搜索替换相应分支。目标函数描述了谓词与真值之间的接近程度，目标函数值反映了执行路径与指定路径之间的距离，亦称为分支距离。

以上面三角形分类程序执行路径 $<s,1,5,9,10,11,12,13,14,e>$ 为例，如果程序执行了输入($a=10$，$b=20$，$c=30$)，程序成功地通过了结点 1 和 5 的错误分支，但在结点 9 却没有能按预期通过正确分支。在结点 9 调用局部搜索，改变程序输入使得程序执行正确分支。一般情况下，如果分支谓词是 a op b，这里 a、b 是算术表达式，op 是关系运算符，相应的目标函数是 f rel 0，表 5-3 给出了谓词表达式与目标函数之间的关系。当谓词取正确分支时，目标函数取负值；反之，取正值。因此要执行谓词的正确分支，需要对目标函数求最小值。例如，在结点 9，针对输入($a=10$，$b=20$，$c=30$)，$f=c-b=10$。源程序必须进行插桩，这样目标函数才可以进行计算，例如，if (eval_obj(9,b,c)) {…}，其中程序中函数 eval_obj 报告在结点 9 时的分支距离。

表 5-3　谓词表达式的目标函数

谓词表达式	目标函数 f	关系(rel)	谓词表达式	目标函数 f	关系(rel)
$a>b$	$b-a$	$<$	$a\leqslant b$	$a-b$	$\leqslant$
$a\geqslant b$	$b-a$	$\leqslant$	$a=b$	$\text{abs}(a-b)$	$=$
$a<b$	$a-b$	$<$	$a\neq b$	$-\text{abs}(a-b)$	$<$

根据目标函数对程序输入进行局部搜索的方法也称为变量替换，依次改变每个变量的值，保持其他变量值不变。变量操作的第一个阶段称为探索阶段，通过增加或减少变量初始

值探索变量邻域。如果某个方向的改变可以提高目标函数值，就可以得到一个模式，从而进入模式阶段，在这个阶段，可以根据这个模式进行更大的改变，继续进行一系列相似的改变，直到相对该变量的目标函数最小值被找到为止。然后，再对下一个变量进行探索。

在图 5-6 所示的例子中，程序执行过程中在结点 9 偏离预期路径，改变变量 a 不影响目标函数，因此首先选择 b，减少 b 会使目标函数值更差，这就需要增加 b 的值，直到 $b>c$。假设 $b=31$，这时新输入向量($a=10$, $b=31$, $c=30$)可以实现在结点 9 按照预期路径执行，但在结点 13 偏离。然后再次调用局部搜索，这次在调整输入变量时需要维持前面的执行路径。新的目标函数是$(a+b)-c$，减少 b 会违反已有的路径约束，我们发现增加 b 可以改善目标函数。最后得到($a=10$, $b=40$, $c=30$)，从而完整路径得到执行。

局部搜索的效果依赖于初始搜索的结果，在图 5-7 所示例子中如果初始输入选择的是($a=10$, $b=10$, $c=10$)，控制流直接进入最后一个结点，然而变量 c 不能变成小于 0，否则，前面路径执行的条件就会被违反，在这种情况下，局部搜索就失败了。

```
void nested_example(int a, int b, int c)
{
    if (a == b)
        if (b == c)
            if (c < 0)
                // target
}
```

图 5-7 局部搜索失效的例子

启发式搜索方法可能进入一个根本无法提高目标函数值的变量值域，这样就会很多无效的搜索和浪费，例如，在图 5-6 中的结点 1 上改变变量 c 的取值就不会产生任何效果。为了提高搜索效率，人们提出利用一些来自程序的附加信息，特别是那些影响当前分支结点的变量信息。例如，在图 5-6 中的结点 5，搜索变量 c 比搜索变量 a 和 b 更好，因为改变变量 a 和 b 可能改变已经成功通过结点 1 的路径条件。

习题 5.6

基于搜索的软件测试主要特点是什么？

5.7 统计测试(Statistics Testing)

1. 概念

统计测试是一种根据定义好的概率分布来随机选择测试用例的软件测试方法，它的有效性来自获得这种定义好的概率分布，这种定义好的概率分布侧重于对应软件中故障分布概率。不同的概率分布对应不同的测试方法，例如，在软件的输入域中按照均匀分布的概率选择测试输入的测试方法，是随机测试；如果按照人们对软件各项功能的使用概率来选择测试输入，就叫操作剖面测试(见 5.8 节)。从这里可以看到，随机测试是一种特殊的统计测试方法，其给定的概率分布是均匀概率。由于人们认为具有揭错能力的测试输入不可能是

均匀分布在输入空间的，因此人们并不认为按照均匀分布概率选择测试输入的随机测试方法是一种非常有效的测试方法。测试用例的选择必须结合待测试软件的模型、功能和结构等信息。

统计测试具有下列两个重要参数：

(1) 测试剖面。即测试用例概率分布，测试用例根据这个分布进行随机选择。

(2) 测试规模。即测试用例数量，决定生成多少测试用例(测试输入)。

这两个参数一般情况下由相应待测试软件的功能和结构测试准则确定。

2. 目标

统计测试在选择测试用例时充分结合了软件功能和结构信息，以确保在按照一定的分布抽样选择测试用例时，尽可能多地执行每个功能和结构元素。不仅可以提高软件测试故障检测能力，同时统计测试的结果还可以为软件可靠性估计提供很好的支持。

3. 方法

设 C 是待测试软件中一个黑盒或白盒测试充分性准则对应的覆盖元素集，Φ 是待测试软件输入域的一个概率分布，任何一个覆盖元素 $c \in C$ 对应该输入分布 Φ 有一个被某些输入执行的概率 p_c，其中最小的概率值为 p，$p=\min\limits_{c\in C}P_c$，$p$ 的大小依赖于不同的待测试软件和覆盖元素集，但 p 越大越好，因为这意味着所有覆盖元素都有很好的机会被执行。

当使用概率方法生成测试用例时，C 中每个覆盖元素 c 被执行的次数是一个随机变量。有两个因素影响 c 被执行的次数，一个是输入概率分布 Φ，另一个是测试规模 N。

输入概率分布 Φ 必须能够允许增加相应的概率，使得那些 C 中最小可能执行元素得到执行。可以有两种方法得到适当的测试输入概率分布，一种是分析方法，另一种是实践方法。分析方法将覆盖元素被激活的条件表示为测试输入的函数，即覆盖元素出现的概率是输入概率的函数，从而最大化最少可能覆盖元素的出现频率(具体过程如图 5-8 和图 5-9 所示的实例)。实践方法需要在软件中插桩，以收集各个覆盖元素的执行次数，一开始采用初始概率分布产生测试用例(可以是均匀分布)，然后逐步细化，直到每个覆盖元素出现频率足够高。

测试规模 N 必须足够大，使得那些最少可能元素在相应的测试输入分布下多次被执行。与测试准则 C 相关的测试质量提供了一个评估测试用例集的理论框架，它的定义如下：

定义 4　一个测试充分准则 C 的覆盖概率是 qN，如果该测试充分准则对应的覆盖元素集 C 中每个元素在 N 次随机输入执行过程中最小的执行概率是 qN。qN 称之为与测试准则 C 相关的测试质量。

p_c 是每一个覆盖元素 $c \in C$ 的执行概率，$p=\min\limits_{c\in C}P_c$ 是下界，测试质量 qN 和测试用例规模 N 可以用这样一个公式连接起来：$(1-p)^N=1-qN$，这个公式的结果是平均每个覆盖元素都要被执行几次。更确切地说，这个公式建立了测试质量 qN 与最少可能元素期望执行次数之间的关系：$n \cong -\ln(1-qN)$。给定测试质量 qN，相应需要的最小测试用例集可以用以下公式计算：

$$N \geqslant \log(1-qN)/\log(1-p) \tag{1}$$

根据公式(1)，依据测试准则设计统计测试用例集的方法分以下两步，其中第一步很关键：

(1) 寻找输入分布，使得 C 中每个元素尽快被执行，这样可以减少测试用例集的规模；或等价地使得输入分布中 p 的值尽可能高。

(2) 评估可以实现相应测试质量 qN 的测试用例集规模 N，从(1)中可以得到 p 的值，然后利用公式(1)计算可得。

由于测试准则对应的覆盖元素与实际故障之间没有非常完美的联系，值得注意的是，测试准则并不影响测试数据的随机生成，它只是作为定义测试输入域和测试规模的一个向导，不需要预先选择一个输入数据子集。概率方法的有效性依赖于一个简单的假设：测试准则提供的信息可以产生一个强化软件错误概率的测试概率分布。在随机测试数据与揭错能力之间有一个直接联系：从公式(1)可知，如果一个错误在每次执行过程中出现的概率不低于 p，那么该错误在 N 次随机测试过程中被发现的概率不低于 qN。

4. 原理

测试准则充分利用待测试软件自身信息作为测试用例选择指南，这些信息来自软件开发过程，或者是结构信息，或者是功能信息，无论哪种软件信息，测试准则都对应一个需要执行的元素集(或叫覆盖元素集)，这里用 C 表示一个测试准则的同时，表示该测试准则对应的覆盖元素集，C 中包含有限个需要运行的元素。例如，分支覆盖准则中包含了所有可执行的边，即 C=｛程序中所有可执行边｝。

给定一个测试准则 C，通常的做法是依据一个确定性方法产生一个测试用例集：测试人员产生一个先验的 N 个测试用例组成的集合，使得 C 中每一个覆盖元素至少被执行一次，通常要最小化这个测试用例集，一般只保证 C 中每一个覆盖元素只执行一次。然而，来自测试准则的一个尖锐的问题是：测试准则与软件错误之间并没有完美的对应关系，不存在软件设计错误的准确模型。测试准则中的每一个覆盖元素被一次或少数几次执行，并不能保证对应的测试用例集具有很高的揭错能力。从这个角度看，任何确定性测试方法的有效性更取决于特定输入的选择，而不是测试准则。

为了改进当前软件测试方法，可以有下列两个不同的研究方向：

(1) 寻找更有针对性的测试准则，对于任何含有故障的软件，无论怎样选择测试用例，该测试准则对应的覆盖元素只要执行一次，就能发现软件中隐藏的错误。

(2) 处理当前测试准则与错误之间关联中存在的问题，通过多次执行测试准则对应的覆盖元素的方法进行补偿。

第一种研究方向是非常艰巨的，因为这种针对性强的测试准则一般完全依赖于残留在软件中的错误，这些错误又因具体的待测试软件的不同而不同，这些信息在测试之前是无法知道的。例如，当前结构测试准则中最严格的路径覆盖准则就不是一种针对性强的测试准则。检查出的错误非常微妙地依赖于执行给定路径的测试用例中很小的一个子集，而且所有路径都执行一次只有对很简单的程序是可行的，任何更细化的准则应该都不现实。

第二个方向由于需要手工选择很多测试用例，应该很乏味。因此需要自动生成测试用例。统计测试因此而产生，统计测试依据一定的测试准则，结合相应的覆盖元素信息，随机生成一个规模较大的测试用例集。

在随机测试中，由于没有关于错误分布和软件结构的任何信息，因此无法判断它的任何分布，例如，操作剖面（即系统投入使用时，输入向量的分布）或者均匀分布，是否是接近最优，因此这意味着随机测试在检测软件故障过程的盲目性和低效性。

白盒测试的有效性取决于具体软件特征和覆盖准则的选择，甚至不少研究发现很多情况下，结构测试不比随机测试好，尽管如此，结构测试还是很多正式的软件测试中必须的一个重要组成部分。

统计测试克服了单纯的随机测试、白盒测试或黑盒测试的不足，充分考虑了软件的结构和功能信息，将它们有机地结合起来。统计测试和随机测试一样也是随机地抽样测试用例，但不同的是统计测试按照一定的概率分布来选择测试用例，这个概率分布满足结构或功能测试的充分性准则（结构或功能的充分性准则在结构或功能测试中被用来生成相应的测试用例，而在统计测试被用来生成测试用例的概率分布）。

统计测试可以看作是随机测试的扩展，它利用功能或结构信息获得概率分布，其意图就是使产生的测试用例可以依据结构和功能等信息合理分布，从而测试效果要比同样规模的随机测试更有针对性和有效性。

统计测试也可看作是结构或功能测试的扩展，它依照一定的概率分布生成测试用例，测试准则对应的覆盖元素可以被多个测试输入执行，这比单纯的结构或功能测试仅仅使用单个测试执行每个覆盖元素更好，单纯的结构或功能测试被认为是确定的，统计测试具有随机性，因为它是依据概率分布随机生成测试用例。

5. 实例

考虑图 5-8 中函数 simpleFunc，函数参数是唯一的测试输入，且输入域是 $D=\{(a,b)\in Z^2: 1\leqslant a\leqslant 50,\ 1\leqslant b\leqslant 20\}$。如果采用随机测试，测试输入参数 a 和 b 是独立地在各自的输入域中随机抽取的，因此当 a 在[1,50]中任意取值时，图 5-8 中第 4 条语句 if(a<=5)取假分支的可能性是取真分支的可能性 9 倍。这种不平衡性意味着要至少生成一个输入检测真分支上潜在的错误时，随机测试会产生更多不必要的执行假分支的输入。在此例中，随机测试产生的测试输入只有在假分支的错误可能性是真分支错误可能性 9 倍以上时才是有效的。结构化测试（白盒测试）选择测试数据执行软件结构元素，例如 simpleFunc 中，选择{(9,2),(4,6),(5,19),(7,18)}分别覆盖图 5-9 中的 c_4、c_3、c_2、c_5，虽然这个测试用例集可以实现路径覆盖，但不一定可以有效地检测错误。例如，图 5-8 第 6 行实现丢掉绝对值：$r=b-19$ 时，现有测试用例就不能发现该错误。

```
1  /* 1 <= a <= 50, 1 <= b <= 20 */
2  int simpleFunc(int a, int b) {
3     int r;
4     if (a<=5) {
5        if (b>=18)
6           r = abs(b-19);
7        else
8           r = b;
9     } else {
10       if (b<=3)
11          r = abs(b-2);
12       else
13          r = 10+b;
14    }
15    return r;
16 }
```

图 5-8 函数 simpleFunc 的代码

$$f(a,b)=\begin{cases}2.94\times 10^{-3} & \text{if } 1\leqslant a\leqslant 5, 1\leqslant b\leqslant 17\\ 1.67\times 10^{-2} & \text{if } 1\leqslant a\leqslant 5, 18\leqslant b\leqslant 20\\ 1.85\times 10^{-3} & \text{if } 6\leqslant a\leqslant 50, 1\leqslant b\leqslant 3\\ 3.27\times 10^{-4} & \text{if } 6\leqslant a\leqslant 50, 4\leqslant b\leqslant 20\end{cases}$$

图 5-9 中函数 simpleFunc 的四个分支 c_2、c_3、c_4、c_5，每个分支被执行的概率分布是 0.25，可以计算出每个输入对的概率分布 $f(a,b)$。例如，函数 $f(a,b)$的第一行，对应分支 c_3，c_3 的定义域 $D_{c_3}=\{1\leqslant a\leqslant 5, 1\leqslant b\leqslant 17\}$，其中包含

了 $5\times17=85$ 个输入对，分支 c_3 的执行概率是 0.25，所以每个输入对的执行概率是 $0.25\div85=2.94\times10^{-3}$。依据概率分布 $f(a,b)$，可以生成统计测试测试用例集对函数 simpleFunc 进行测试。

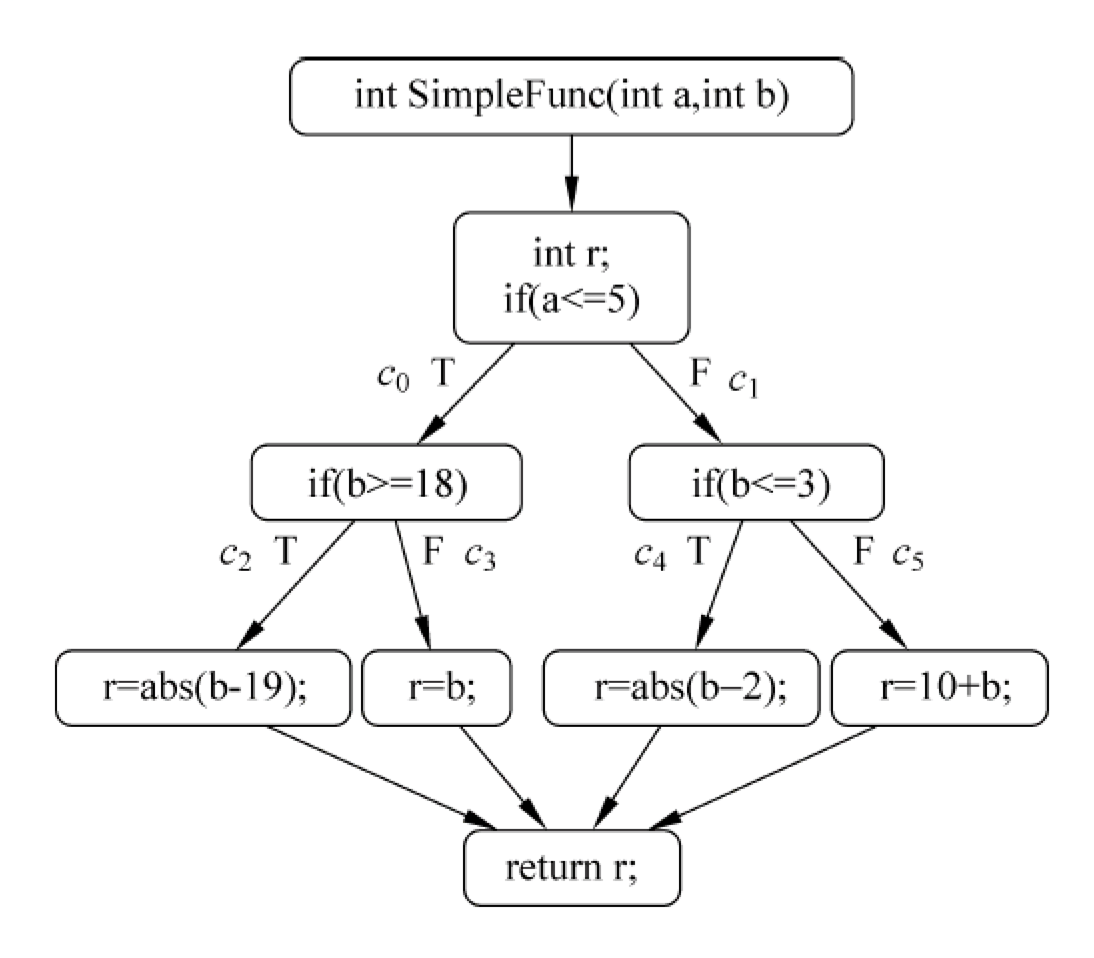


图 5-9　函数 simpleFunc 的控制流图

6. 优缺点

统计测试将随机测试和结构或功能测试方法结合起来，利用软件功能和结构信息产生的概率分布随机选择测试输入，可以有效提高错误检测能力。一方面它克服随机测试几乎不考虑待测软件任何信息的盲目性，另一方面又克服了功能和结构测试方法的相对确定性，统计测试在没有精确错误模型的前提下是一种非常实用的软件故障检测方法。因此统计测试可以实现功能或结构测试，同时也是一种随机测试方法。

统计测试的不足在于它的有效性受到测试输入概率分布的影响，而一个合理的概率分布的获得具有一定的难度。而且统计测试需要执行大量的测试用例，其测试成本要远高于一般的功能和结构测试。

习题 5.7

统计测试方法具有什么特点？

5.8　基于操作剖面的测试（Operational Profile Based Testing）

基于操作剖面的测试是一种统计测试方法，它按照人们对软件各项功能的使用概率来选择测试用例。

1. 概念

软件操作剖面（Software Operational Profile，SOP）是一个软件的操作（功能）的集合，且每个操作对应一个出现概率。用 E 表示操作集合，P 概率函数，对于操作集合中任何操作元素 e，$e\in E$，$P(e)$表示该操作出现概率或使用频率，它的值一般是人工统计的或通过估

计得到的。利用操作剖面的信息进行软件测试的方法，称之为基于操作剖面的测试。

2. 目标

在测试过程中首先找出出现频率高的错误（在使用频率高的操作或功能中包含的错误，其出现频率也高），即基本上按照错误出现频率高低的顺序去分配测试资源，尽早发现错误。

3. 方法

软件的操作剖面是软件如何被使用的一个定量描述，因此它是对软件进行准确和充分测试的一个前提条件，是软件测试的一个向导，特别是在资源受限的条件下，利用操作剖面可以保证对最常用的操作投入更多的测试资源，从而实施科学有效的测试。

4. 优缺点

操作剖面的获取需要一定的成本，而且获得的操作剖面不一定准确，这在一定程度上降低了基于操作剖面的软件测试方法的有效性。

5. 研究现状

研究利用操作剖面分配测试用例，进行测试用例约简、软件质量及可靠性估计等。

习题 5.8

基于操作剖面的测试有什么优缺点？

5.9　变异测试（Mutation Testing）

1. 概念

变异测试通过在程序中植入一些程序员容易犯的错误，或利用相关的启发式方法植入一些错误，这些错误都是通过对原来的程序做一些句法上的修改，例如，把（$i<0$）改为（$i\leqslant 0$），从而形成一组新的程序，称之为变异体，通过运行针对原有程序的测试用例集，如果这些原来的测试用例能够发现变异体中错误，称之为杀死变异体，否则称之为变异体存活。最后通过分析被杀死的变异体数量和存活的变异体数量，来推测软件程序中可能仍然潜在的错误数量，评估针对原有程序的测试用例集的测试充分性程度。如果一个测试用例集能杀死所有植入错误的变异体，那么这个测试用例集就有可能有能力检测出所有可能潜在的错误。

2. 目标

变异测试是一种基于错误的软件测试方法，通过变异测试可以得到变异充分性得分，这个变异充分性得分一方面可以作为软件质量评估的依据，另一方面，可以作为测试用例集测试充分性和有效性的度量（或者说错误检测能力的度量）。通过变异测试可以帮助设计产生充分的测试用例，有效地检测软件中潜在的错误。

3. 原理

为了说明变异测试的原理，这里考虑一个例子，计算一个桶中玻璃球的数目。在一个很大的桶中装着很多大小和重量都一样的玻璃球，现在要估算桶中玻璃球的数目。要求不能使用数球的方法或者称球的方法，只能利用手上现有的 100 个大小和重量都与桶中彩球都一样，而颜色不一样的黑球。科学的估算彩球个数的方法是首先将 100 个黑球放入桶中，尽量将这些球混合均匀，然后从桶中随机抓取 100 个球，统计黑球的个数 n，这样桶中彩球的个数 N 可以通过计算公式$\frac{N}{100}=\frac{100-n}{n}$得到。例如，在随机抓取到的 100 个彩球中有黑球 10 个，由此可以估计桶中原来有彩球 $N=900$ 个。

现在我们把一个程序 P 看成一个桶中，设已经设计的测试用例集 T 对其进行了充分测试，未能发现新的错误，但其中仍然可能潜在很多错误，这些可能包含的错误可以看成是其中放置的玻璃球，通过分别在程序 P 的 100 个拷贝中植入 100 个错误，这样就形成了 100 个变异体，对这 100 个变异体运行测试用例集 T，如果能在其中 20 个变异体中发现错误，相似地就可以估算出程序 P 中包含的潜在错误大约是 400 个。

在估计桶中玻璃球数量的例子中，新加入的 100 个黑球需要满足一个重要条件，即这些黑球与桶中其他球不同之处只是颜色，而大小和重量必须相同。这些差别必须不能影响人们下一轮选球，例如，如果这些新加入的黑球是金属球，那么就会明显影响人们随机选球的结果，这样利用这种机制估计桶中玻璃球数量就不科学了。同样，在软件程序中植入错误也要具有一定的限制，否则，通过植入错误的方式来评估软件中可能存在的错误是不科学的。

由于程序中潜在的错误数量是不可知的，而且程序的变异体数量庞大(一般是程序规模的平方)，人们不可能测试所有的变异体。在实际的变异测试中，只测试很小一部分变异体，利用这很小部分的变异体来模拟全部可能的错误。变异测试的科学性基于以下两点假设：第一点是假设程序员是很成熟的(Competition Programmer Hypothesis)，写出来的程序基本上接近正确的程序，程序中存在的错误只是一些很小的错误，只要通过简单的句法修改就可以改正。因此，基于这个假设，变异测试中的变异体都是对源程序经过简单的句法修改得到的；第二点假设是程序中错误的耦合效应(Coupling Effect)，即程序中复杂错误是由简单错误耦合形成的，如果一个简单错误是由正确程序经过一步简单句法修改形成的，那么复杂错误往往是正确程序经过多步简单句法修改形成的。因此，如果一个测试用例能够检测出一个简单错误，那么，一般也极有可能检测出由多个简单错误耦合形成的复杂错误。

4. 方法

图 5-10 描述了变异测试的一般过程，一开始原始程序 P，针对 P 按照一定的测试标准或要求设计了一个测试用例集 T，当对 P 运行这个测试用例集 T，如果发现错误，则修改错误，并在此运行测试用例集 T 中的测试用例，以确保修改的正确性和没有引入新的错误。知道运行 T 不再发现 P 中的错误。然后，通过对 P 的每个拷贝逐个进行微小的修改，产生变异体 P'(例如，将 P 中某个“与”操作符换成 P'中“或”操作符)。这样产生一组变异体。

变异体产生规则称之为变异操作，有很多变异操作，表 5-4 中只例举了一些常用的变异操作。

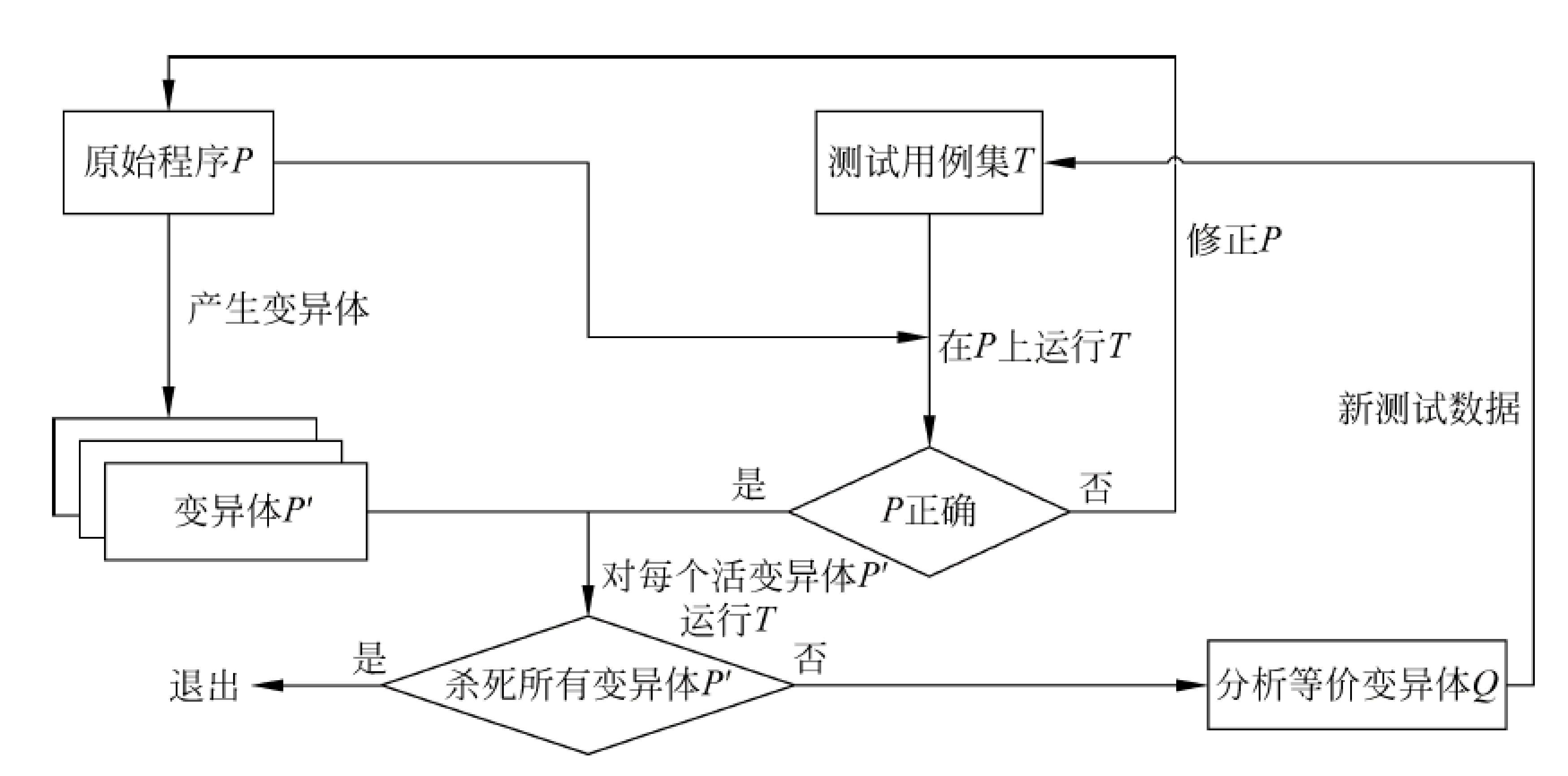


图 5-10 变异测试的一般过程

表 5-4 常用的变异操作

变异类别	错误模型	例子
常数变异	不正确的常数	x＝x＋1；变异为：x＝x＋3；
运算符变异	不正确的运算符	x＋＋；变异为：＋＋x；if(x＜y)变异为：if(x＜＝y)
语句变异	语句放置错误	z＝x＋1；变异为：break；或将该语句删除
变量变异	变量使用错误	z＝x＋1；变异为：z＝y＋1；

针对每一个变异体 P' 运行测试用例集 T，如果运行结果与相应的 P 的运行结果不同，则变异体被杀死，即测试用例集 T 可以发现 P 中植入的错误。否则，该变异体存活了下来。当针对所有的变异体运行完测试用例集 T 之后，如果没有发现存活的变异体，则变异测试结束，可以认为测试用例集 T 具有较好的测试充分性。否则，通过增加测试用例的方式改进测试用例集 T，使其能够杀死存活下来的变异体。当然，其中存在一些等价的变异永远都无法被杀死，这已经成为变异测试中的一个难题，有待进一步研究。最后，我们可以从变异测试结果中得到变异充分性得分，是被杀死的变异体个数与所有非等价变异体总数之比。变异测试的一个目标就是要使得变异充分性得分为 1，这样就使得测试用例集可以充分检查所有可能的程序错误，包括植入的错误。

由于程序的变异体很多，对每一个变异体运行测试用例集 T 是一件非常耗时耗力的事情。因此，人们提出两种方法来改进变异测试过程：一个称为弱变异(Weak Mutation)，该方法在针对某个变异体运行测试用例集 T 时，任何时候，只要发现与源程序 P 结果不同，就认为该变异体被杀死，停止执行剩下的测试用例，开始选择下一个变异体重新执行 T；另一方法称为抽样变异，该方法首先产生出所有可能的变异，然后每次从众多的变异操作中随机选择一个产生一个变异体，执行测试用例集 T。可以根据测试需求和可用资源状况，决定抽样变异的次数，但研究表明，只要抽样出其中 10%以上的变异体就可以与完全执行所有变异体的效果相差不大。以上两种方法可以结合起来使用。

习题 5.9

变异测试的特点是什么？

5.10 冒烟测试(Smoke Testing)

1. 概念

冒烟测试广泛应用于管道修理、木质乐器修理、电子、计算机软件开发和舞台测试等领域。通过冒烟测试检测产品在通过修理之后或首次装配之后不会导致重大或灾难性的后果，如修理后的管道不再泄露、乐器键盘已正确安装、电路不会短路、软件不会崩溃、舞台可以承受更大压力等。

Smoke Test 最初源自微软以及很多大的软件公司，"daily build and smoke test"——即每天自动编译及 Smoke Test，这里的 Smoke Test 仅仅是一个简单的测试，看看我们编译好的产品是否"冒烟"以检查每天编译的结果是否成功。

2. 目标与意义

(1) 能最小化集成风险。项目组可能遇到的一个很大的风险是，项目组成员根据不同的系统功能各自开发不同的代码，但是当这些代码集成为一个系统的时候，也许系统完成不了预期的功能。这种风险的发生取决于项目中的这种不兼容性多久才被发现，由于程序界面已经发生了变化，或者系统的主要部分已经被重新设计和重新实现了，相应的排错工作将非常困难和耗时。极端情况下，集成的错误可能会导致项目被取消。每日构造和冒烟测试可以使这种集成错误变得非常小，而且便于解决，防止了很多集成问题的产生。

(2) 能减小产品低质量的风险。这种风险是和集成不成功、集成出错相关联的。每天对集成的代码做一些少量的冒烟测试，即可杜绝项目中那些基本的质量问题。通过这种方式，使系统达到一种周知的良好状态，维护这样的系统可以防止系统逐步恶化到耗费大量时间排查质量问题的地步。

(3) 能简单化错误诊断。当系统每天都进行 build 和测试时，系统任何一天发生的错误都能够变得十分精细，便于排查。比如在 17 日系统还运行正常，18 日就出错了，那么只需要检查这两次 build 之间的代码变化就可以了。

(4) 能极大地鼓舞项目组的士气。看到产品的不断成长，能够极大地鼓舞项目组的士气，有时甚至不管这个产品到底用来做什么。开发人员可能会为系统显示了一个矩形而感到激动。通过每日构造，产品每天进步一点点，保证项目士气的持续高涨。

习题 5.10

什么叫冒烟测试？它的主要作用是什么？

5.11　基于性质的软件测试方法(Property Based Testing)

1. 概念

从理论上来说,软件测试应该是完全的。所谓完全,指的是所用的测试用例应该覆盖所有可能的软件输入、软件所处环境的各种变化情况(如硬件故障)等。但由于软件的行为空间和输入集很可能是无穷大的,即使不是无穷的,也有可能大到无法在有限的时间内完成。实际上,人们对软件的某些部分的行为较为感兴趣,也就是说,"完全"的测试用例的某个子集更加值得测试,也应该被优先测试。基于此,有必要对软件进行选择性的测试。

基于性质的软件测试是一种测试的方法论,用以解决如何有效地进行选择性测试的问题。在测试前,会选定一个或几个性质,在测试过程中,与选定性质无关的软件行为将被忽略,从而达到减少测试用例集合大小的,降低测试的时间、人力成本开销的目的。被选定的性质通常是软件中的重要或关键性质,它们最值得被测试。例如,当使用一个文本编辑软件时,人们常常更加关注输入的信息的安全性,希望当出现忘记保存、系统崩溃、计算机断电等突发情况时,仍然能够保证已键入或已修改的信息能够被成功保存下来,这个时候,软件在自动保存、自动恢复等方面的性质就更值得测试;相比之下,其他一些功能,如调整文本格式、页面布局等或许没那么重要:它们的不完善会给用户带来不便,但至少不会出现灾难性的后果。

2. 原理

基于性质的测试方法是一种一致性测试(Conformance Testing),它的目的在于确认(Validate)而不是证明(Verify)软件符合某些规约。正式的证明也许需要建立一个模型,并用形式化的方式来证明;或者,它需要穷尽程序执行过程中的所有可能的路径。基于性质的测试与之不同,它通常假设待测程序满足给定性质,围绕着给定性质设计测试用例,通过测试来观察确认待测程序是否满足给定性质。如果在测试过程中给定性质被破坏了,我们就可以认为待测程序不满足给定性质;否则,在足够多的测试用例被用于测试,并满足一定的覆盖度要求,我们可以认为待测程序符合给定性质。注意程序符合给定性质的要求与程序没有缺陷是不同的,一个有缺陷的程序可能没有破坏给定性质。一个发现了软件破坏了既定性质的测试应该是成功的测试,如果没有发现破坏性质的行为,并且测试用例被证明是完全的,或者虽然不完全但是满足一定的覆盖度要求,那么,这样的测试也应该是成功的。

3. 方法

在基于性质的测试里,选定的性质往往用一定的形式表述出来,可以是某种形式化的语言,可以是某些公式,甚至可以是自然语言——只要它可以在测试过程中发挥指导作用。一般地,选定的性质会被描述成一些规约(Specification)的集合,这些规约表明了测试人员期望待测软件的行为符合的特征,这些特征可能仅仅是外部的,但也可能是内部实现相关联。性质的规约越详尽越正式,和程序的实现联系越紧密,越有利于有效指导测试过程。理想情况下,一个性质应该用形式化的规约描述,这样无论是测试用例生成还是测试结果分析,都

能更加严谨地进行；再者，使用形式化的规约来描述性质有利于自动化程度的提高。然而，很多情况下，使用形式化的描述方法会遇到很多困难，如不够灵活、可用的形式化语言表达能力不够强、可读性差、成本较高等等，因此如何有效地描述性质（包括形式化和非形式化）仍然是研究的重点。

在测试中，性质主要指导以下几个方面：软件的哪些部分应该被测试；设计什么样的测试用例；测试到什么时候可以终止；什么样的测试是成功的测试。基于性质的测试里，使用的测试用例应该紧密围绕着给定性质设计生成，有针对性地对给定性质进行详尽的测试，使用形式化的规约描述的性质有可能自动生成测试用例和测试预言（Test Oracle），而且能够生成更加严谨的测试用例和预言。有时候，当使用了一组测试用例进行测试后，软件的性质没有被破坏，但是进入一种“危险状态”，即在继续进行测试的情况下有可能出现破坏性质的行为，这个时候就应该使用更多的测试用例，以期能够要么确认这种危险状态不会导致破坏性质的行为，要么发现破坏性质的行为。例如，在一个登录（Log-in）系统中，很重要的一点就是它的安全性，即不能提供正确的ID和密码的用户是不允许登录的。假设系统能满足这一点，但是系统会准确地提醒用户无法登录的原因是ID的错误还是密码的错误，抑或是两者错误，是否提供了非法用户获取合法用户的ID和密码的可能性，这就是一种危险状态，需要进一步测试。

基于性质的测试方法可以是黑盒的，也可以是结合了白盒和黑盒的。它可以在代码层面用来测试一个程序，也可以在更高层面上测试一个复杂系统的行为。复杂的系统通常会作某种程度的抽象以便于测试，如用于基于模型检测的测试中。可见，基于性质的测试不仅仅是在测试的某个阶段使用，而是可以在很多阶段使用。从理论上来讲，基于性质的思想可以贯穿测试的整个过程，只不过在测试难度大、更加需要选择性测试的测试阶段运用这种方法更有价值。

习题 5.11

1. 简述基于性质的软件测试方法的特点。
2. 结合个人实践，给出一个利用程序性质进行测试的具体例子。

5.12 极限测试（Extreme Testing）

极限编程（XP）是一种相当新的软件开发过程，可以使开发人员快速地产生高质量的代码，这里质量是代码对其规格说明的满足程度。XP重视采取简单的设计、在开发人员和客户之间建立联系、不断地测试代码库、重构以适应规格说明的变更，以及寻求用户的反馈。这种方法适合于中小规模的软件开发，这些软件的规格说明的变更非常频繁，甚至需要接近实时的沟通。

XP与传统的开发过程相比有几处不同：首先，它避免了大型项目的综合症，即在开始编码之前客户与编程小组交流、设计软件的每个细节，其实这种做法存在重要缺陷，因为为了反映新的业务准则或市场情况，客户的规格说明和需求必须进行不停地变更；其次，XP方法可以避免编写不重要的功能，将精力集中在必须的功能之上，有助于在短时间内开发出

高质量的软件；最后，XP 方法最重要的不同之处是将精力集中在测试上，在经历了一个非常全面的设计阶段之后，传统的软件开发模型会建议首先编码，然后再进行测试，但在 XP 中，必须首先生成单元测试用例，然后编写代码通过测试。

XP 开发模型用以下 12 个核心实践来驱动开发过程：

(1) 计划与需求分析。将市场和业务开发人员集中起来，共同确认每个软件特征的最大商业价值；以使用场景的形式重新编写每个重要的软件特征；程序员估计完成每个使用场景的时间；客户根据估计时间和商业价值选择软件的功能特征。

(2) 小规模、递增地发布。努力添加细微的、实在的、可增值的特征，频繁发布新版本。

(3) 系统隐喻。编程小组确认隐喻，便于建立命名规则和程序流程。

(4) 简要设计。实现最简单的设计，使代码通过单元测试。由于变更不断发生，因此不要在设计上花太多时间，只是不停地实现。

(5) 连续测试。在编写模块之前就生成单元测试用例；模块只有在通过单元测试之后才告完成。此外，程序只有在通过了所有的单元测试和验收测试完成之后才算结束。

(6) 重构。清理和调整代码库。单元测试有助于确保在此过程中不破坏程序的功能；应在任何重构之后重新进行所有的单元测试。

(7) 结对编程。两位程序员协同工作，在同一台机器开发代码库，这样可以对代码库进行实时检查，能极大地提高缺陷的发现率和纠正率。

(8) 代码的集体所有权。所有代码归全体程序员所有，没有哪一个程序员只致力于某一个代码库。

(9) 持续集成。每天在变更通过单元测试之后将其集成到代码库。

(10) 每周 40 小时工作。不允许加班，如果每周全力工作 40 小时，就不需要加班。在重大发布前的一星期除外。

(11) 客户在现场。开发人员和编程小组可以随时接触客户，这样可以快速、准确地解决问题，使开发过程不至于中断。

(12) 按标准编码。所有的代码看上去必须一致，设计一个系统隐喻有助于满足该原则。

以上 12 个核心实践可以归纳为以下 4 个概念：

- 聆听客户和其他程序员的谈话；
- 与客户合作，开发应用程序的规格说明和测试用例；
- 结对编程；
- 测试代码库。

1. 概念

先创建单元(模块)测试和验收测试，然后再创建代码库，这种形式的测试就叫极限测试，或者叫测试驱动的开发。

单元测试是极限测试中采用的主要测试方法，它的原则是：所有代码模块在编码开始之前必须设计、生成好单元测试用例，在产品发布前须通过单元测试。这种首先设计单元测试的做法是 XP 方法的亮点，可以迫使开发人员在开始编码之前理解规格说明，排除混淆。单元测试的测试用例与编写的软件具有一样的价值，应当进行很好的保存。

验收测试是XP方法中第二类、也是同等重要的极限测试类型，验收测试的目的是判断应用程序是否满足如功能性和易用性等需求。在设计或计划阶段，由开发人员和客户来设计验收测试。验收测试可以是人工的，也可以是自动化的，发现的错误集中提交开发小组，开发小组根据错误的优先级修正软件，然后重新进行验收测试。需要注意的是，程序可能通过单元测试，但不一定能通过验收测试。

2. 优点

(1) 可以使人们对代码满足规格说明具有信心。
(2) 在开始编码前，就展现了代码的最终结果。
(3) 更好地理解了应用程序的规格说明和需求。
(4) 可以先实现一些简单的设计，稍后再放心地重构代码以改善程序的性能，而无须担心破坏应用程序的规格说明。

3. 实例

开发一个命令行的应用程序，接受一个正整数 $n(0<n<1000)$，判断 n 是否为素数。如果 n 是素数，程序应返回信息，说明其为素数；如果 n 不是素数，程序也返回信息，说明其不是素数；如果 n 不是一个有效输入，程序应显示一条帮助信息。针对该应用程序的极限测试用例如表5-5所示。

表5-5 极限测试用例表

用例编号	输入	预期输出	注释
1	$n=3$	n 为素数	对有效素数的测试
2	$n=1000$	不是	对边界范围内的测试
3	$n=0$	不是	对不为素数情况的测试
4	$n=-1$	帮助信息	对低于下边界情况的测试
5	$n=1001$	帮助信息	对高于上边界情况的测试
6	两个输入	帮助信息	对输入值的正确个数的测试
7	$n=$“a”	帮助信息	对输入为整数而非字符类型的测试
8	n 为空格	帮助信息	对输入值为空时的测试

习题 5.12

简述极限测试的概念。

5.13 模糊测试(Fuzzing Testing)

1. 概念

模糊测试是一种通过提供非预期的输入并监视异常结果来发现软件故障的方法，模糊测试一般是一个自动或半自动的过程，这个过程包括反复操纵目标软件并为其提供处理数据。模糊测试中生成测试输入的工具称为模糊器，所有的模糊器可分为两大类：基于变异

的模糊器，这种模糊器对已有数据样本应用变异技术创建测试用例；基于生成的模糊器，这种模糊器通过对目标协议或文件格式建模的方法从头开始生成测试用例。

2. 目标

模糊测试的基本思想是自动产生和发送大量随机的或经过变异的输入值给软件，如果发生失效或异常，便可挖掘出软件系统存在的薄弱环节和安全漏洞，这种方法已经发展成为对软件质量有深远影响的测试技术。

3. 方法

模糊测试方法的选择依赖不同的因素，可能有很大的变化，没有一种绝对正确的模糊测试方法。模糊测试方法的选择完全取决于目标应用程序、研究者的技能，以及需要测试的数据所采用的格式。然而，无论要对什么进行测试，也不论确定选择了哪种方法，模糊测试总要经历几个基本阶段：

(1) 识别目标。在没有考虑清楚目标应用程序的情况下，不可能对模糊测试工具或技术做出选择。如果是在安全审核的过程中对内部开发的应用程序进行模糊测试，目标应用程序的选择应该小心谨慎。相反，如果对第三方应用程序进行安全漏洞的发掘研究，这种选择就有一定的灵活性。在识别目标应用程序的时候，需要考查开发商过去被发现的安全漏洞的相关历史。这可以通过浏览安全漏洞收集站点来完成，例如，securityfocus 或 secunia。一个开发商如果在过去的安全漏洞历史记录方面表现不佳，那么很可能有比较差的编码习惯，最终将会导致更多的安全漏洞被进一步发现。选择了目标程序之后，还必须选择应用程序中具体的目标文件或库。如果需要选择目标文件或库，应该选择那些被多个应用程序共享的库，因为这些库的用户群体较大，出现安全漏洞的风险也相应较高。

(2) 识别输入。几乎所有可被人利用的漏洞都是因为应用程序接受了用户的输入并且在处理输入数据时没有首先清除非法数据或执行确认例程。枚举输入向量对模糊测试的成功至关重要。未能定位可能的输入源或预期的输入值对模糊测试将产生严重的限制。任何从客户端发往目标应用程序的输入都应该被认为是输入向量。这些输入包括消息头、文件名、环境变量等，所有这些输入向量都应该是可能的模糊测试变量。

(3) 生成模糊测试数据。一旦识别出输入向量，模糊测试就可以生成。如何使用预先确定的值、如何变异已有的数据或动态生成数据，这些决策将取决于目标应用及其数据格式，不管选择了哪种方法，这个过程应该引入自动化。

(4) 执行模糊测试数据。这一步，模糊测试成为动词，执行过程可能包括发送数据包给目标应用程序、打开一个文件或发起一个目标进程。同样，这个过程的自动化也至关重要。

(5) 监视异常。在模糊测试过程中，一个至关重要但却被经常忽视的步骤是对故障或异常的监视过程。例如，如果我们没有办法确切地指出是哪一个数据包引起崩溃的话，那么向目标 Web 服务器发送 10 000 个模糊数据包并最终导致服务器崩溃便是一次无用的努力。监视可以采取多种形式并且应该不依赖目标应用程序和所选择的模糊测试类型。

(6) 确定可利用性。一旦故障被识别，因审核的目标不同，需要确定所发现的 Bug 是否可以被进一步利用。这是一个典型的人工过程，需要具备安全领域的专业知识，因此执行这一步的人可能不是最初执行模糊测试的人。

模糊测试可以与其他测试技术结合使用，例如，在符号执行和约束求解过程中使用模糊测试技术，或者以代码覆盖率为导向，进行演化的模糊测试方法。

4. 优缺点

一方面，人们认为应用模糊测试只能发现一些简单的错误，因为软件测试问题是非常复杂的，模糊测试只是关注其中人们感兴趣的部分，并采用简化的不完全测试方法。而且模糊测试的代码覆盖率一般都很低，例如，如果输入包含了校验部分，那么在模糊测试过程中可能执行了校验部分的代码。代码覆盖率经常被用来估计模糊器工作的效果，但只是作为模糊器质量的一个指导性的指标，而且每个模糊器都是用来发现一类不同的软件故障。

另一方面，模糊测试发现的故障有时候非常严重，黑客可以利用这些发现的漏洞来实现攻击，事实上，模糊测试已经成为黑客攻击已有软件的常用的重要方法和技术。模糊测试的输入一般是随机产生的，用这种方式产生的输入很难覆盖边界值，这可能是模糊测试的缺点，但模糊测试强调软件的安全性和可靠性，它经常能发现一些出乎意料的软件故障，这些故障是测试人员很难通过测试用例的设计来发现的。

习题 5.13

模糊测试的特点是什么？

5.14 自适应测试(Adaptive Testing)

自适应测试也称为软件测试的控制论方法。

1. 概念

软件控制论是探讨软件工程理论和控制理论相互渗透的交叉理论，该理论通过将软件问题归结为控制问题，以及将控制问题归结为软件问题，研究这两个领域的互补和结合，从而达到分别发展这两个领域的作用。软件测试的控制论方法是软件控制论的重要组成部分，它把软件测试的问题归结为控制问题，被测试软件当作被控制对象，软件测试策略当作相应的控制器，这样被测试软件和测试策略就构成一个闭环反馈控制系统。通过控制理论指导软件测试过程，从而希望达到科学有效地测试软件的目的。

2. 目标

软件测试的控制论方法，即适应性软件测试，可以利用测试过程中的历史信息指导未来的测试步骤(如测试用例的选择)，利用这些测试信息估计被测试对象的性质和参数，根据这些参数可以更有针对性地选择测试用例，实现给定目标下的最优测试(如图 5-11 所示)。既可以提高软件测试效率，实现以最小代价发现最多故障，又可以提高检测性能的稳定性，提高软件可靠性估计的精度。软件测试的完全自动化和效率最大化要求，是推动软件测试控制论方法发展的重要动力。

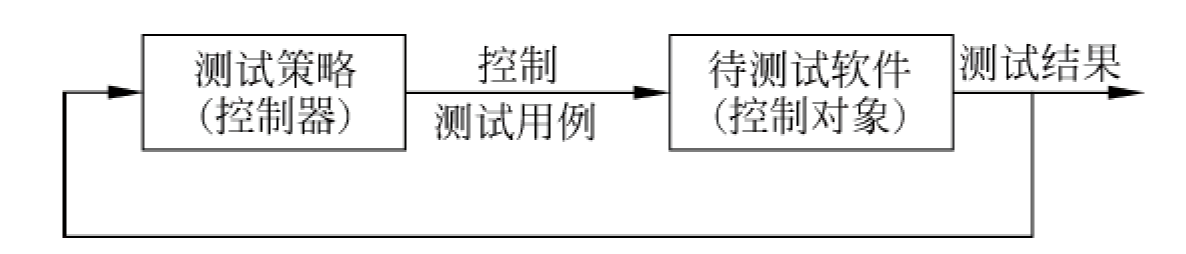


图 5-11 软件测试作为一个控制问题

3. 原理

自适应控制系统是通过在线实时了解被控对象,不断调节控制器,使系统的性能达到技术要求或最优。自适应系统有 3 大要素:一是在线实时地了解对象;二是有一个可调环节;三是使系统性能达到要求或最优。自适应控制可以分为直接自适应控制和间接自适应控制。在间接自适应控制中,被控对象的参数未知,首先在线估计对象参数,利用估计值对控制器参数进行调整使系统性能指标达到要求(如图 5-12 所示);而在直接自适应控制中,不对对象参数进行估计,直接通过调整控制器参数来改进系统性能(如图 5-11 所示)。

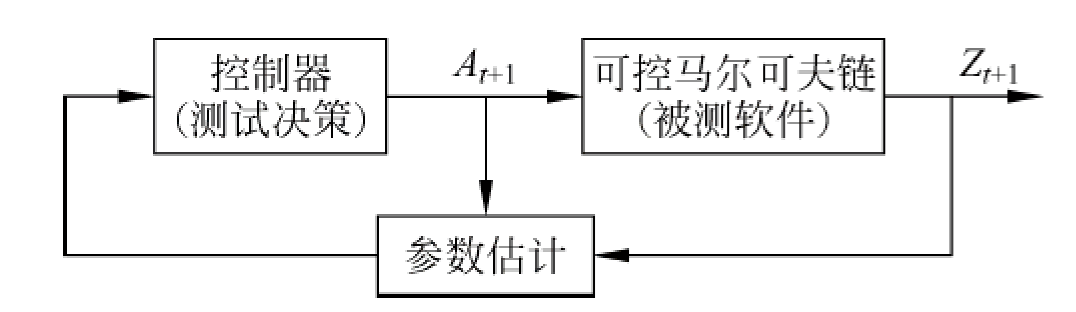


图 5-12 自适应测试系统的结构框图

自适应测试的理论基础是以自适应控制系统为基础,即可控马尔可夫链。以软件测试为例,自适应测试方法把被测软件当作控制对象,利用可控马尔可夫链理论设计和优化软件测试策略,并把测试策略作为控制器和被测软件构成一个闭环反馈系统,它是软件测试的控制论方法的具体实现。

因此,适应性软件测试对应于软件测试过程中的适应性控制,意味着软件测试策略应该根据测试过程中收集到的测试数据以及人们对待测试软件的理解进行在线调整。与路径测试、功能测试等传统方法不同的是:适应性软件测试具有明确的优化目标,在测试的过程中不断根据先验知识进行目标优化,而传统方法一般都不具备动态优化功能。

4. 方法

自适应测试方法是一种利用反馈和自适应控制原理的软件测试方法,如图 5-13 所示。自适应测试策略具有两个反馈环:第一个反馈环包括:待测试软件(控制对象)、测试历史数据库 H_t(测试用例和相应的测试结果 Z_{t+1})和测试策略(控制器),在这个反馈环中,测试历史数据库 H_t 用来根据测试策略(控制器)产生下一条测试数据,此时测试策略(控制器)可以是某些测试数据充分性准则,如代码覆盖准则、路径覆盖准则等;第二个反馈环包括待测试软件(控制对象)、测试历史数据库 H_t、参数估计和测试策略(控制器),在这个反馈环中,测试历史数据库 H_t(测试用例和相应的测试结果 Z_{t+1})和已有的测试策略组成的测试历史,既用作改进和产生新的测试策略,也用作进行一些参数估计(包括测试用例集的检错能力、错误率的估计),例如,将基于均匀分布的随机测试改进为非均匀分布的随机测试,或者将数据流测试修改为边界值测试等,主要原因是在测试过程中,人们对于待测试软件的认识以及测试效果的认知不断加强。

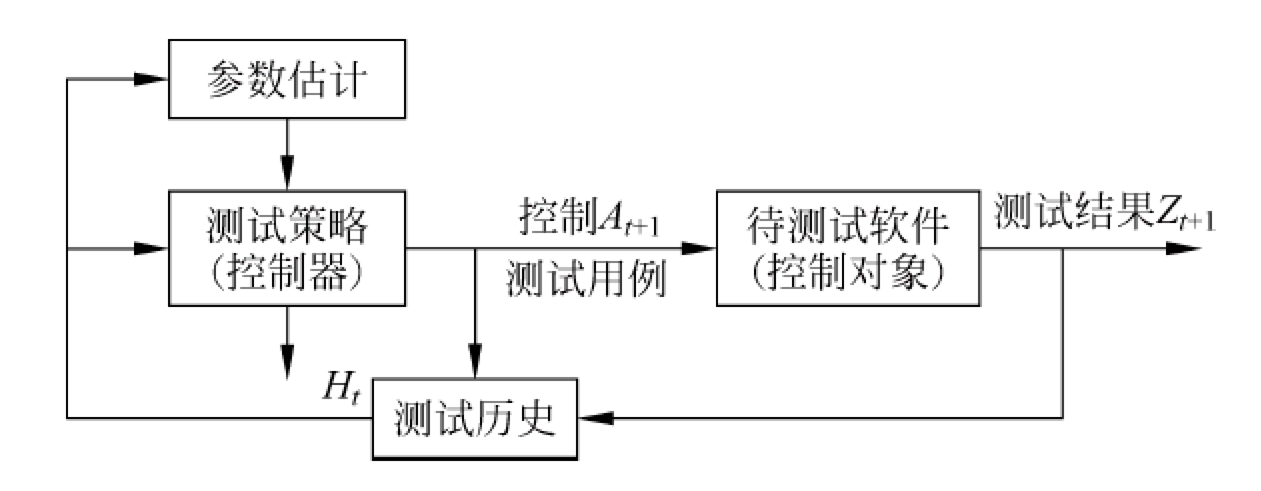


图 5-13 更具体的自适应测试方法图

在测试过程中,包括软件可靠性增长测试,软件变异测试和软件可靠性评估过程,针对软件模型的参数未知,需通过在线估计参数,再利用估计值调整控制器,即采用间接自适应控制方法进行软件测试,形成所谓的软件测试的自适应策略。其中,根据可控马尔可夫链模型,一些常用参数意义如下:

$Z_t^{(l)}=\begin{cases}1, & \text{在 } t \text{ 时刻检测到软件系统 } l \text{ 中缺陷}\\ 0, & \text{在 } t \text{ 时刻没有检测到软件系统 } l \text{ 中缺陷}\end{cases}$ 表示是否检测到缺陷;

N 为软件总缺陷数;

$Y_t=j$;$j=0,1,2,\cdots,N$;$t=0,1,2,\cdots$表示 t 时刻被测软件含有 j 个缺陷;

A_t 是测试过程每一步应采取的决策;

θ_i 是第 i 种决策的测试决策缺陷检测率;

t 时刻采取策略 A_t 的代价定义为 $W_{Y_t}(A_t)$,而

$$W_{Y_t}(A_t=i)=\begin{cases}w(i), & \text{if } Y_t=j\neq 0\\ 0, & \text{if } Y_t=0\end{cases}$$

由控制器方程,在软件测试过程中,每一步应该选取的测试决策应满足:

$$A_t=\arg\min\left\{\frac{w(i)}{\theta_i}\right\}$$

如果被测软件参数 N 和 θ_i 已知,就可直接根据上述公式求得测试决策,从而逐步把测试进行下去,这种测试成为最优测试;而由于实际测试中,被测软件参数的不可知性,这就需要利用自适应测试,在每步测试中通过在线估计系统参数(N 和 θ_i),再利用估计值求得最优测试决策,逐步把测试进行下去,直到剔除所有缺陷。随机测试则是在任意一步测试中,从已知的决策集$\{A_t\}$中随机选取决策,这样选中任何一个决策的概率都是均等的,而不存在最优决策,这样把测试进行下去,直到剔除所有的缺陷。

可控马尔可夫链由五元素模型描述:$\langle S,A,\{A(y)|y\in S\},Q,W\rangle$,其中各元素的含义为:$S$、$A$ 分别是系统所有可能的状态所组成的状态空间集和决策集;$\{A(y)|y\in S\}$是系统状态为 y 时的决策;$Q(B|y,a)=Pr\{Y_{t+1}\in B|Y_t=y,A_t=a\}$,$B\subset S$ 表示在状态 $Y_t=y$时,采用决策 $A_t=a$ 转移到状态 Y_{t+1}的概率;W 是每步测试代价。该模型是一个在 $t=0,1,2,\cdots$时刻上随机可控的系统。如果假设在 t 时刻系统状态为 $Y_t=y\in S$,而此时的决策为 $A_t=a\in A(y)$,那么将会有以下事件发生:系统在转移方程 Q 的决定下,以代价 $W_{Y_t}(a)$转移到 Y_{t+1}状态。一旦发生转移,系统进入新的状态,并会选择新的决策继续循环决策过程。

以下是依据可控马尔科夫链模型的自适应测试过程:

(1) 根据可控马尔可夫链模型,得到以下方程:

$$v(j)=\min_{1\leqslant i\leqslant m}\left\{\frac{w_j(i)}{j\theta_i}+v(j-1)\right\};\quad j=2,3,\cdots,N$$

表示当软件系统缺陷数(状态)为 j 时,最优决策 i(即控制器输出)的选取应该使得 $w_j(i)/\theta_i$ 最小;当 $w_j(i)=w(i)$ 时,最优控制器为:$\min\left\{\frac{w(i)}{\theta_i}\right\}$。

(2) 由控制器方程,在软件测试过程中,第 t 步选取的测试决策应满足:$A_t=\arg\min\{v(j)\}=a_t$;这里 $t=1,2,\cdots$

(3) 如果被测软件参数 N 和 θ_i 已知,就可直接根据上述公式求得测试决策,从而逐步把测试进行下去。

(4) 利用自适应测试,在第 t 步测试中通过求解最优化问题:$\min\sum_{j=-1}^{t}\left[z_j-\left(N-\sum_{i=-1}^{j-1}z_i\right)\theta_{a_j}\right]$,在线估计系统参数($N$ 和 θ_i)。

(5) 利用估计值求得下一步最优测试决策,$A_{t+1}=\arg\min\{v(j)\}=a_{t+1}$,逐步把测试进行下去,直到剔除所有缺陷。

以上过程中把测试用例或测试对象输入域划分为多个等价类,用可控马尔可夫链模型建模软件测试过程,在测试过程中不断把历史测试信息反馈给测试过程,使得随着测试过程的进行,可以逐渐了解被测对象的性质和测试用例的能力,并通过自适应算法调整模型参数并确定下一步最有效的测试用例等价类,在该等价类中选择一个测试用例作为被测对象的输入。

图 5-14 描述了待测试软件的状态转换行为,其中状态 $j(0\leqslant j\leqslant N)$表示待测试软件中还隐藏有 j 个缺陷,在时刻 t,如果对状态 j 应用测试决策 A_t,待测试软件仍然具有 j 个缺陷的概率是:$1-j\theta_{A_t}$,转移到具有 $j-1$ 个缺陷的概率是:$j\theta_{A_t}$。

5. 优缺点

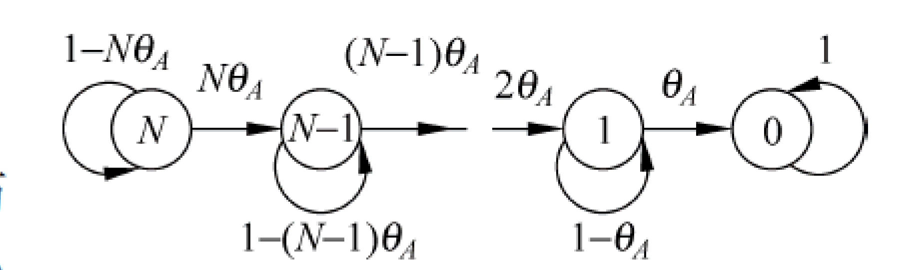


图 5-14 可控马尔可夫链的状态转移图

现有测试技术存在两个问题:没有引入测试历史信息,大量的测试信息(比如,测试用例的有效性)没有用于指导下一步的测试行为;没有对被测对象的参数和对象的性质做在线估计。由此,产生的结果是:缺乏对测试过程的改进,不能得到最有效的测试过程;对测试对象的性质不了解,难以实现给定的测试目标(比如,测试代价最小、可靠性评估的方差最小)。而自适应测试,正是利用测试过程中的历史信息,并用它来指导未来的测试步骤(比如,测试用例的选择),同时用这些测试信息估计被测对象的性质和参数,根据这些参数可以更有针对性的选取测试用例,实现给定目标下的最优测试。

但自适应软件测试方法的效果依赖于反馈系统的形式化和反馈机制的质量,目前相应的反馈机制还远未成熟,尽管反馈机制在软件过程随处可见,但高度形式化和定量化的反馈机制还未形成,这大大制约了适应性软件测试方法的应用。

6. 补充阅读

计算机适应性考试(computer adaptive testing,CAT)

根据考生对试题的反应情况，试题库适应性地选择下一道试题。例如，考生对一个中等性问题回答得很好，计算机就会给他选择一个高级一点的问题；如果考生对这个中等性问题回答得不好，下一个问题就会是一个初等层次的问题。测试停止在可以准确反映考生水平的层次上。

当考生通过计算机进行考试时，计算机可以通过不断估计考生的能力来选择下一道考题，在拥有适当的题库，并且考生能力存在很大差别的前提下，CAT 比传统的纸笔考试更加有效。

传统的纸笔考试是一种固定形式的考试，所有的考生都回答试卷中统一的考题。这些题目对有些考生来说，有的可能很难，有的可能很容易，都会给考生增加一些多少不等的分数。但是这个分数一般不能真实反映学生的真实水平，因此，试题和考生都需要一定程度的精度。

在计算机适应性考试过程中，考生的能力可以不断地进行评估，考题根据评估结果不断进行调整，这样 CAT 可以为考生选择适当的试题，从而从考生的反映中最大限度地获得考生能力的信息，考生一般不会分配到太容易或太难的考题，这个选择过程可以克服手工选择可能存在的错误，提高测试精度。

习题 5.14

1. 软件测试的控制论方法与传统测试方法有哪些区别和联系？
2. 自适应考试可以看成是一种自适应控制在考试中的应用，请模仿图 5-11、图 5-12 或图 5-13 画出相应的反馈机制图，并给出简要的解释。

5.15 导向性随机测试(Concolic Testing)

1. 概念

导向性随机测试是一种将具体执行与符号执行相结合的自动化测试方法，其目标是通过生成测试输入来执行程序中的所有可行路径，以发现程序缺陷。其中，Concolic 是将具体(CONCrete)和符号(symbOLIC)两个词结合在一起所构成的一个新词(CONCrete＋SymbOLIC＝ Concolic)。具体执行是以具体的输入值执行程序；符号执行使用变量符号代替具体的输入值，模拟执行程序，得到程序各路径条件的符号化表达式，然后对符号表达式进行约束求解，以得到能够执行指定程序路径的具体输入值。而导向性随机测试则是将具体执行与符号执行相结合的测试技术，其思想是在具体执行过程中收集路径条件的符号表达式，并按照深度优先的策略将路径条件逐一取反，然后通过约束求解的方式求解取反后的路径条件，以获得执行新路径的测试输入。当不能求解路径条件时，使用具体的值来代替符号表达式，以简化路径条件。

2. 方法

导向性随机测试的过程描述如下：

(1) 选择程序 C 的输入变量作为符号执行中的符号处理。

(2) 对目标程序 C 进行静态插桩，每个桩记录路径执行的符号条件。

(3) 利用给定的输入运行插桩后的目标程序(在一开始,程序的输入随机产生)。

(4) 获得程序 C 在具体给定输入下的实际执行路径的符号公式 φ_i。

(5) 将执行路径 φ_i 上的一个分支条件取反,产生一个新的路径符号公式 ψ_i。

(6) 利用约束求解器求解新路径的符号公式 ψ_i,得到目标程序 C 的下一个具体输入。

(7) 跳转到第(3)步,直到所有可行路径都被执行。

3. 实例

考虑测试图 5-15 中的程序,从程序中可以看出,通过随机生成测试用例测试到 Error 分支是很困难的,概率非常小。以下说明导向性随机测试能克服这个困难。

(1) 选择测试输入变量 a、b、c。

(2) 对目标程序进行静态插桩,如图 5-16 所示,每个分支都标注了执行的符号条件。

(3) 首先随机产生一个输入(0,0,0)。

(4) 输入(0,0,0),执行路径的符号公式是 a!=1。

(5) 将符号公式 a!=1 取反,得到新的符号公式 a=1。

```
// Test input a, b, c
void f(int a, int b, int c) {
  if (a == 1) {
    if (b == 2) {
      if (c == 3*a + b) {
        Error();
} } } }
```

图 5-15　被测试程序

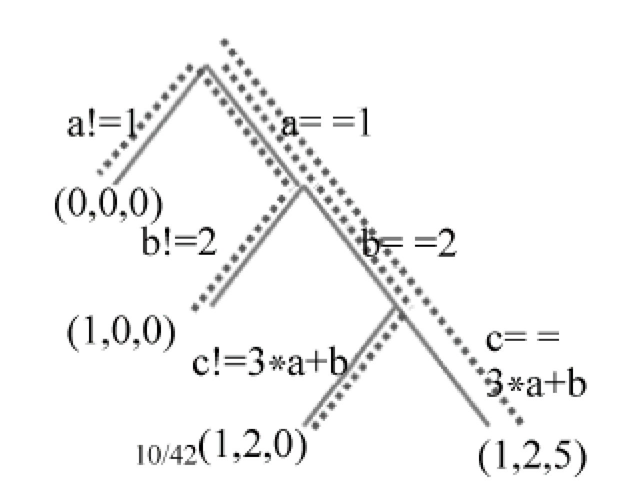


图 5-16　导向性随机测试示意图

(6) 根据新的符号公式 a=1,产生新的输入(1,0,0)。

(7) (转第(3)步,第 1 轮循环开始) 利用新输入(1,0,0),执行目标程序。

(8) 得到新输入(1,0,0),执行路径的符号条件公式为 a=1 且 b!=2。

(9) 将符号公式 b!=2 取反,得到新的符号公式 a=1 且 b=2。

(10) 根据新的符号公式 a=1 且 b=2,产生新的输入(1,2,0)。

(11) (转第(3)步,第 2 轮循环开始)利用新输入(1,2,0),执行目标程序。

(12) 得到新输入(1,2,0),执行路径的符号条件公式为 a=1 且 b=2 且 c!=3a+b。

(13) 将符号公式 c!=3a+b 取反,得到新的符号公式 a=1 且 b=2 且 c=3a+b。

(14) 根据新的符号公式 a=1 且 b=2 且 c=3a+b,产生新的输入(1,2,5)。

(15) (转第(3)步,第 3 轮循环开始)利用新输入(1,2,5)执行目标程序。

(16) 得到新输入(1,2,5),执行路径的符号条件公式为 a=1 且 b=2 且 c=3a+b。

(17) 此时已经无法通过对某个分支条件进行取反得到新的符号公式,测试结束。

习题 5.15

查阅符号执行(Symbolic Execution)方面的资料,了解符号执行方面的理论、方法、工具和应用。

5.16 图形用户界面测试(GUI Testing)

图形用户界面,通常称为GUI(Graphical User Interface),是计算机技术的重大进展之一,它已经越来越成为用户喜欢的一种人机接口界面,它几乎是无处不在的。GUI的好坏将直接影响用户使用软件时的效率及心情,也直接影响用户对所使用的系统的印象。开发者用专门的、越来越大的模块来实现它们。图形用户界面的代码量构成高达60%的应用程序的总代码量。因此,制定GUI测试的测试原则,掌握GUI测试的测试流程,研究它的测试用例生成方法,开发高质量的GUI自动化测试软件是非常有意义的。这对减少软件开发的成本,提高软件质量是非常有帮助的。

1. 概念

用户图形界面(又称图形用户接口)是指采用图形方式显示的计算机操作用户接口。与早期计算机使用的命令行界面相比,图形界面对于用户来说在视觉上更易于接受。WIMP是图形界面电脑所采用的界面典范,W指窗口(Window)、I指图标(Icon)、M指菜单(Menu)、P指指标(Pointer)。一个GUI是一个分层的图形化的软件前端,它从一个固定的事件集中接受用户产生的和系统产生的事件,并且产生确定的图形输出。一个GUI包含许多图形对象,每个图形对象有一个固定的属性集合。在GUI执行的任何时刻,这些属性有着离散的值,这些值的集合组成了GUI的状态。

软件界面的图形化已成为当今软件的一个重要特征,图形化的用户界面不仅越来越普遍,也越来越复杂,图形用户界面测试技术也因此越来越受到重视,人们已经在针对图形用户界面的测试用例生成、测试预言和测试充分性准则等方面做了很多、很系统的研究。

图形用户界面测试是一种软件测试过程,测试的对象是软件产品的图形用户界面,目的是确保图形用户界面符合用户要求或规格说明书要求。测试的方法是通过设计、执行各种高质量测试用例,系统地覆盖系统功能,充分执行所有图形用户界面。为此,需要处理好两个方面的问题:一个是测试域范围的确定,另一个是测试序列的确定。

测试域范围的确定就是确定测试哪些图形用户接口操作。跟命令行系统不同的是,具有图形用户界面的系统有很多需要测试的操作对象,例如,一个很小的Microsoft WordPad就有325个可能的图形用户接口操作,一些大的软件系统,图形用户接口操作很可能还要多一个数量级。

测试序列的确定问题是指某些系统功能的执行必须按照一个复杂的图形用户接口事件序列来完成。例如,打开一个文件,用户必须首先单击文件菜单,选择“打开”操作,使用对话框选择要打开的文件,按“确定”按钮,最后查看新打开的窗口。因此,当测试的功能越来越多时,这些操作序列就会迅速增长,这给测试用例的设计提出了很严峻的挑战。

在图形用户界面的测试中,回归测试也是一个重要的问题,当图形用户界面在不同版本中发生变化后,原来的测试用例往往需要作相应的变化,因为原来的按钮、菜单或对话框发生了变化。为了克服图形用户界面测试中的问题,除了自动化方法之外,人们提出了很多GUI测试方法,例如有限自动机模型方法,把图形界面系统抽象为一个有限自动机来进行测试,但对于复杂的系统,自动机也会变得非常复杂而不是非常好用。

2．目标

GUI 测试与软件测试中的其他方面的测试有很大的区别，其目标主要体现在解决以下几个方面的问题：

（1）一个 GUI 的可能接口空间是非常庞大的。每个 GUI 活动序列可能会导致系统处在不同的状态上。一般来说，GUI 活动的结果在系统不同的状态上是不同的。这样就必须在一个庞大的状态集上进行 GUI 测试，这个工作很难完成，即使是借助自动化测试。

（2）GUI 的事件驱动特性。由于用户可能会单击屏幕上任何一个像素，因此对 GUI 来说可能会产生非常非常多的用户输入。而要让计算机自动地模拟这些输入是困难的。

（3）界面的美学具有很大的主观性。界面元素的默认大小、元素间的组合及排列次序、界面元素的位置、界面颜色等，这些对不同的用户来说可能会有不同的结果。因此，如何才能代表大部分客户的意见也是界面测试的一个难点。尤其是测试人员和开发人员可能会有不同的意见。

（4）应用程序中的功能通过图形用户接口进行调用，糟糕的设计会使得界面与功能混杂在一起，这使得界面的修改可能会导致更多的错误，增加了测试的难度和工作量。

3．原理

GUI 的重要特征主要包括图形化、事件驱动、具有等级体系结构、包含各种对象且这些对象具有各种性质。因此，GUI 可以定义为：具有等级的图形化软件前端，可以接受用户产生的或者系统产生的事件，这些事件来自确定的事件集合，作为软件的输入，并对应确定的输出。一个 GUI 包括一组图形对象；每个对象都有一组性质，在 GUI 执行的任何时候，这些性质都具有一组离散的值，这些离散的值构成 GUI 的状态。

虽然 GUI 测试有其特殊的地方，但其测试过程与传统方法一样，主要包括以下步骤：

（1）根据 GUI 测试覆盖标准，确定每一个用户界面事件为测试对象，以确保其正常工作。

（2）针对每个测试对象产生 GUI 测试输入序列，包括鼠标点击、菜单选择和对象操作等。

（3）产生 GUI 测试预期输出，包括屏幕的显示、窗口的位置和标题等。

（4）执行测试用例，比对测试结果与预期结果，判断测试是否结束等。

测试充分性准则是一组决定测试什么时候停止的准则，比较常用的有结构化覆盖准则，如语句覆盖、分支覆盖等。GUI 测试不能照搬，因为 GUI 的测试对象是大量的事件和事件序列，不能用基于代码的测试准则来衡量 GUI 测试，因此人们定义了基于事件的覆盖准则，不仅考虑所有可能的事件，还覆盖事件之间的交互，考虑事件流序列的长度等因素。

用户界面测试是一个需要综合用户心理、界面开发设计技术的测试活动。尤其需要把握一些界面设计的原则，遵循一些设计的要点来进行测试。根据界面设计的原则来制定一份界面设计规范，这份界面设计规范需要得到项目组全体人员的认可，作为设计界面和测试的依据，也是开发人员开发界面和修改界面的依据。下面介绍的是 IBM 的用户界面架构规范，可以参考这些原则和规范来制定适合自己的测试项目的界面规范。IBM 用户界面架构，简称 UIA（User Interface Architecture），提出了 12 个方面的界面设计原则：

(1) 亲和力(Affinity)。通过良好的外观设计,让界面更具有亲和力。亲和力的设计是整个设计过程的主体部分。界面的设计应该简易朴素,建立层次感,设计一个显示出良好的供给能力的视觉方案。

(2) 协助(Assistance)。提供主动的协助。软件系统应该帮助用户执行各种各样的任务。不同用户的系统知识和处理任务的能力水平不同,软件系统要能识别用户的能力,给予用户适当的帮助。

(3) 有效(Availability)。让用户可以在任何时候、以任何次序在同一个视图使用所有的对象。尽量避免使用模态窗口,模态窗口会禁止其他窗口的操作,这样会限制用户与系统交互的能力,导致很多不便。

(4) 鼓励(Encouragement)。让动作可以预见并可以恢复。鼓励用户探索系统,尝试操作,查看结果,撤销或删除操作。在操作之前,让用户知道执行操作会带来的后果。

(5) 熟悉(Familiarity)。让用户基于已有的关于软件系统的知识来使用新的系统。统一的视觉设计和界面交互技巧能减少用户学习使用新软件或新版本的时间,同时降低学习的难度。

(6) 明显(Obviousness)。让对象和控件是明显、直觉、显而易见的,类似它们在现实世界的样子。使用图像或文字提示来明确对象和控件的功能,易于用户识别。

(7) 个性化(Personalization)。允许用户对界面进行个性化设置,允许用户按个人的需要和想法裁剪界面。让用户把经常使用的或是感兴趣的功能放到快捷的位置,这样可以节约时间、提高用户的使用效率及满意程度。

(8) 安全(Safety)。不要让用户轻易接触到危险的操作,同时提供给用户足够的帮助信息,减少用户犯错的机会。例如,Windows 操作系统会隐藏平时用户用不到的关键的系统文件,避免用户不小心删除掉它们,造成不必要的麻烦。

(9) 满意(Satisfaction)。让用户感觉到操作流畅。迅速显示用户操作的结果,任何加在用户任务上的延迟都会影响用户对系统的信心。及时的反馈可以让用户判断是否对操作结果满意,如果不满意,则可以撤销操作,或是更换。

(10) 简单(Simplicity)。尽量减少界面上的对象和动作的个数,但是能足以让用户完成任务。

(11) 支持(Support)。让用户控制系统,让用户自己定义完成任务的过程。不要把自己认为正确的做事方式强加给用户,而限制了用户可能的选择。

(12) 多样性(Versatility)。支持替代的交互方式。没有一个唯一的交互方式是在任何情况下都是最好的。提供给用户多种选择,让用户在不同环境下选择最佳的解决方法。

4. 方法

手工测试即通过人工的方法将测试用例输入计算机并对应用程序进行测试,来检测其功能是否符合需求说明书的要求。随着计算机软件技术的不断发展,软件中的 GUI 越来越复杂,需要进行测试的 GUI 事件序列十分庞大,用人工的方法进行测试,效率会比较低下,而且往往会有许多重复的工作,给测试人员带来很多麻烦。但是,手工测试也有其不可替代的地方,主要包括以下几点:在测试用例的设计中测试人员的经验和对错误的猜测能力是工具不可替代的;对图形界面的体验中人类的审美观和心理体验是工具不能模拟的;检查

正确性时人们对是非的判断、逻辑推理的能力是工具不具备的。

自动化测试是软件测试发展的一个必然结果。软件测试的一个显著特点是重复性，重复让人产生厌倦的心理，重复使工作量倍增，因此自动化测试成为了重要的手段。自动化测试可以明显减少整个软件开发周期中软件测试的时间和资金的开销。自动化可以保证测试有规律而且一致地进行，在早期实现错误检测，从而提高质量，缩短产品上市时间。

目前广泛使用的生成GUI测试脚本的方法是捕获/回放技术。捕获/回放技术即测试者通过点击鼠标和键盘输入，与GUI进行的交互工作。用脚本记录这些事件序列，然后以自动测试的方式对事件进行回放。这种方法产生的脚本是固定的，不能灵活地改动。如果在测试时需要不同的输入，测试脚本需要重新生成。如果GUI部件经过了测试对有问题的部分进行了修改，在进行回归测试时也不能使用原来的测试脚本，给测试者带来不便。另外手动录制测试脚本的时候容易出错，捕获/回放过程有可能会记录下很多错误的或是冗余的点击鼠标事件和键盘输入。

自动化测试是对手工测试的一种补充，自动化测试不可能完全替代手工测试。手工测试、自动化测试，一个都不能少。关键是在合适的地方使用合适的测试手段。

常见的自动化测试软件有：IBM Rational测试工具，它可以完成测试软件开发周期中所有阶段的测试，在GUI方面经常用到的是RationalRobot和Visual Test；WinRunner是Mercury公司开发的工具，可用于基于Windows或是Web的应用程序的测试，可以使用外部的数据库测试应用；SilkTest是Segue公司的GUI测试工具，它运行于Windows平台，主要用于测试MFC库生成的对象和GUI部件。

大部分的开放源码的GUI测试工具是针对Java开发的GUI软件的。常见的有：Abbot能够对Java应用进行GUI单元测试和功能测试；GUITAR主要采用基于事件的工具和技术，并包含一个测试用例生成器；qftestJUI它是在UNIX系统上使用的，测试带有Java/Swing应用的GUI部件。

习题 5.16

选择一个小型软件，如计算器、QQ等，对其进行GUI测试，对本节提供的理论与方法进行验证和总结。

5.17　随机测试(Random Testing)

1. 概念

在待测试软件所有可能的输入中，随机选择和产生测试输入并对待测试软件进行测试，这种测试方法称为随机测试。随机测试是一种最为简单的测试方法，它几乎不利用任何关于软件的额外信息(除了软件输入信息)，也不需要测试人员的任何经验和技能。

2. 目标

随机测试方法是那些依据测试充分性准则，需要通过精心设计产生测试用例的测试方法的一个重要补充，可以弥补由于人们考虑不周或考虑有偏向造成的不良影响，经常能发现人们

意料之外的软件故障。因此,随机测试已经被推荐为软件测试过程中最不可缺少的一步。

3. 原理

Myers 认为随机测试是一种最弱的测试方法,这种测试方法利用的测试用例集在错误检测能力方面几乎不可能达到最优,或者接近最优,他建议人们使用更聪明的测试方法,而最好不要用随机测试方法。

而人们认为随机测试方法简单易行,容易自动化,是一种不可忽视的重要的软件测试方法。特别是很多实践研究证明,随机测试方法与其他所谓更聪明的方法(如等价划分测试、路径覆盖测试方法)相比,其错误检测能力不差,有时甚至更好一点,而这些方法的使用成本明显要比随机方法昂贵很多。

4. 方法

随机测试方法只利用待测试软件的输入模型,该输入模型定义了待测试软件的所有可能的输入,不需要对所有的测试输入进行划分,测试输入的选择可以根据预期输入的分布情况随机选择,而如果没有关于输入分布的相关信息,只需利用均匀分布,随机产生测试输入。

5. 实例

考虑测试一个可以将平面直角坐标系的坐标(x,y)转换为极坐标系坐标(r,H)程序,其中 $r=\sqrt{x^2+y^2}$,$\cos H=x/r$。直角坐标系和极坐标的原点重回,极坐标极轴与直角坐标的 x 轴重合,极角是逆时针角度。

输入:

x:$[-320,320]$,最小增量 $1/2^6$

y:$[-240,240]$,最小增量 $1/2^7$

输出:

r:$[0,400]$,最小增量 $1/2^6$

H:$[0,(2\times \mathrm{pi}-1/2^6)]$,最小增量 $1/2^7$

对于这个程序的输入分布没有任何特别信息,因此,可以利用均匀分布,随机选择 x 和 y 的值(x 可选值有 40 961 个,y 有可选值 61 441 个)。要慎重使用其他分布,因为这些分布使用不当会导致有些输入域无法被测试到。

随机测试可以手工完成,也可以在完全没有人干预的情况下自动完成,自动随机测试具有最好的成本有效性,然而,要获得完全的自动化就必须解决 3 个方面的问题,即测试输入的自动生成、预期结果的自动生成和测试输出的自动检查。

只要待测试软件的输入模型建立好,利用伪随机数生成器随机生成测试输入并不困难。然而,预期输出的自动生成或自动检查存在很多问题,且很多情况下不大可行,除了以下情况:利用另一个独立的可信的软件执行相同功能,产生预期输出;测试过程中只考虑软件系统是否崩溃;软件输出易于检查,例如,排序程序的输出就是一个排好序的数组;软件的输入根据输出易于产生,例如,只要将平方根程序的输出进行平方就可以产生它的输入。

上例中,坐标转换程序的预期输出是否正确可以利用反函数进行检查,利用两种坐标之间存在关系 $r\sin H=y$,$r\cos H=x$ 可以在一定范围内判断程序输出是否正确。

即使随机测试的完全自动化存在很多问题，但随机测试仍然被看作一种非常有效的测试方法，因为随机测试的测试用例生成不需要任何成本。随机测试没有定义任何覆盖度量，它的一个完成准则可以定义为在某个置信区间内（如95%），发生不一致输出的概率（如0.001）。对于一些输入比较复杂的程序或者一些符号输入的程序，需要一些专门的方法对输入进行处理，以使得待测试软件可以方便地使用随机测试方法。

6. 优缺点

- 优点　测试输入随机生成，几乎不要什么附加信息，成本低，易于自动化；随机测试能够产生人们无法产生的测试用例，从而发现一般的软件测试难以发现的软件故障；随机测试的结果可以为软件可靠性估计提供有效数据。
- 缺点　没有定义任何覆盖标准作为测试目标，可能存在很多没有测试到的地方。

习题 5.17

编写程序，实现对本节实例中的直角坐标和极坐标互相转化应用程序进行自动的随机测试，通过实践总结随机测试的优点、缺点和需要解决的问题。

5.18 自适应随机测试（Adaptive Random Testing）

1. 概念

自适应随机测试是一种改进的随机测试方法，旨在提高随机测试方法的性能，该测试方法基于人们在测试过程中建立起来的一种广泛认可的直觉和经验：一个测试用例如果没有发现软件故障，那么另一个相近或相似的测试用例可能也不会发现软件故障，因此当运行一个测试，而这个测试用例没有发现任何故障时，在选择下一个测试用例时，要尽可能选择距离该测试用例远一点的测试。自适应随机测试方法强调：在随机选择测试用例的过程中，要使得测试用例尽可能均匀地分散开，这样随机产生的测试用例才有可能有更多机会发现软件故障。

2. 目标

任何随机测试可以使用的地方一般都可以使用自适应随机测试方法进行改进，提高测试用例的分散性，从而尽早地发现错误。

3. 原理

为了提高随机测试的错误检测能力，自适应随机测试利用已有测试用例的信息，特别是测试过程中未能发现故障的测试用例信息来指导生成新的测试用例。自适应随机测试基于人们的一种观察，即引发系统故障的输入一般都是串联在一起形成一个或多个连续的区域。相应地，如果一个测试用例和一个没有发现错误的测试用例靠得很近，就很可能仍然发现不了错误；相反，如果这个测试用例离这个未发现错误测试用例距离很远，则这个测试用例发现错误的可能性会更高。因此，自适应随机测试强调测试用例选择的分散性，距离度量是该方法的一个关键。

4. 方法

这里主要介绍固定候选用例集大小的适应性随机测试，一种最典型、最简单、也是应用较广泛的方法。

固定候选用例集大小的适应性随机测试使用 2 个测试用例集：①S——用于存放已用来测试程序但并没有引发程序失效的测试用例，$S=\{s_1,s_2,\cdots,s_n\}$；②C——用来临时保存用于产生新的测试用例的候选测试用例，大小固定为 k，$C=\{c_1, c_2,\cdots,c_k\}$。下一个测试用例将从 C 中按照如下步骤产生：

(1) 计算每个候选用例 $c_i\in C$ 与成功用例集 S 之间的距离 d_i，d_i 为 c_i 与每个成功用例 $s_j\in S$ 之间的距离的最小值。

(2) 选择 d_i 最大的候选用例 c_i 为下一个测试用例，即选出距离成功用例最远的候选用例。

5. 实例

对于一维的输入域[0,10]，其中的失效域为[7,8]，对于随机测试方法来说，用 F-度量（F-度量是指找到第一个导致程序失效时所用测试用例的个数）的方法测试可得其找到失效时的期望值为 10；对于基于距离的适应性随机测试来说，初始 S 集合为空，为最大限度地覆盖输入域，C 集合初始为{0,1,2,⋯,10}，按照固定候选用例集大小的适应性随机测试方法，则从 C 集合中依次产出下一个测试用例的顺序为(0,10,5,2,7,1,3,4,6,8,9)，因此在第 5 次进行测试候选用例时，便测试出了失效状态，F 值为 5，效率比随机测试提升了很多。

6. 优缺点

很多实验研究表明，自适应随机测试方法可以有效提高随机测试效率，特别是 F-度量的值，即更快地找到软件中潜在的错误。但有些研究人员认为自适应随机测试耗费大量的计算资源，在实际测试中有时不像预期的那样，因为对于有些测试场景，自适应随机测试不一定能够做到尽量将测试用例分布开。

习题 5.18

对于二维的输入域[(0,0),(10,10)]，其中的失效域为[(7,7),(8,8)]。对于随机测试方法来说，用 F-度量的方法测试可得其找到失效时的期望值是多少？对于基于距离的适应性随机测试来说，设初始 S 集合为空，为最大限度地覆盖输入域，C 集合初始为{0,1,2,⋯,10}，按照固定候选用例集大小的适应性随机测试方法，则从 C 集合中依次产出下一个测试用例的顺序是什么？此时 F 值是多少？

5.19 反随机测试(Anti-random Testing)

随机测试是一种黑盒测试，既不利用软件的结构信息，又不利用已有测试用例的执行信息。与功能测试等其他黑盒测试相比，随机测试的测试用例产生的特点是它的随机性，其他

黑盒测试技术的测试用例设计都具有一定的确定性。因此，随机测试可以给人们对于软件的质量从一个很重要的角度提供了信心。但同时人们也发现，对于软件中一类非常难以发现的错误，依靠随机测试，也很容易造成疏漏。

反随机测试同样是一种黑盒测试技术，该技术首先为待测试软件随机生成一个测试用例，随后每一个测试用例的生成都要求与已生成的测试用例保持最大的距离。反随机测试虽然不利用软件的结构信息，但追求测试用例之间要尽量分散开。反随机测试的测试用例除了第1个测试用例是随机生成的，其他测试用例都具有一定的确定性。

为了说明测试用例间距离的概念，设待测试软件有 n 个数值输入，每个测试用例是一个 n 维向量，设 $A=(a_1,a_2,\cdots,a_n)$ 和 $B=(b_1,b_2,\cdots,b_n)$ 是该待测试软件的两个测试用例(严格来说，只是测试输入，测试用例还要包括相应的预期输出)。测试用例间距离可以有多种定义方法，常用的定义有海明距离和欧几里得距离2种：

(1) A 和 B 间的海明距离。$H(A,B)=$ A 和 B 间相应分量不同的个数，例如 $A=(1,2,1,2)$，$B=(2,1,2,2)$时，由于这两个向量之间有三个分量都不一样，因此，$H(A,B)=3$。

(2) A 和 B 间的欧几里得距离。$H(A,B)=\sqrt{(1-2)^2+(2-1)^2+(1-2)^2+(2-2)^2}=\sqrt{3}$。

反随机测试方法的具体步骤分为以下3步：

(1) 给待测试软件的 n 个输入变量中每一个输入变量随机指定一个值，这样形成一个随机的测试用例。

(2) 产生一个新的测试用例，使得这个测试用例与已有的测试用例保持最大的总距离，并把该测试用例加入已有测试用例集。

(3) 重复第(2)步，指导所有可能的输入被遍历，或者产生了指定数量的测试用例。

反随机测试的目标有两个：首先，是对于那些无法穷尽测试的大系统，反随机测试希望通过测试用例的最大限度的分散，尽可能多地发现软件故障；其次，即使对于小型的可以穷尽测试系统，反随机测试希望尽可能早地发现隐藏其中的故障。

习题 5.19

反随机测试的目标是什么？

5.20　结对测试(Pair Testing)

结对测试是一种由两个测试人员结对在一起对待测软件进行测试的方法，一个测试人员控制鼠标和键盘，另一个测试人员做记录，一起讨论测试场景，提问题。结对的两个测试人员一般情况下一个是专业测试人员，另一个是开发人员或业务分析员。结对测试不是一种非常正式的测试，它是一种探索性测试方法。探索性测试是一种非正式测试技术，该技术需要丰富的测试经验和对待测试软件良好的了解。结对测试与结对编程相似，而结对编程在极限编程中得到成功的实践。结对测试方法可以充分发挥两个人的智慧、实现知识技能的互补和共享，以及相互激励，是个体测试方法的一种有效补充。

结对测试可以让测试人员获得对待测试软件的良好理解，并能学会如何发现和排除软件故障。该方法适合于需求和规格说明不是非常清楚，测试小组新成立不久，需要在短时间

内了解待测试软件的情况。特别是，由开发人员和测试人员结对测试时，他们对待测试软件的不同视角会给该软件更全面的测试，他们分别对于软件本身和测试方法的知识会互相分享、相互补充，从而产生很好的测试效果。而且两个人在一起工作可以相互促进、取长补短，最重要的是两个人工作在一起非常有趣。对于发现那些难以再现的错误，以及需要人工判断测试执行是否正确的情况，结对测试是两双眼睛总会比一双眼睛好。但结对测试不是万能的，与其他测试方法一样，结对测试存在风险，它有可能会失败。结对测试只有在测试人员相互尊重相互信任的环境下才会有效，否则，会存在很多问题。而且，如果只做结构化测试，结对测试就不适合，因为白盒测试只要一个人按照要求去做就可以了。

总之，结对测试是将合作与测试工作结合在一起，可以实现测试人员与开发人员之间以及测试人员相互之间的交流和知识共享，有利于培训新员工、打破工作人员之间的隔阂，但结对测试只有在合适的场合用好了才能有效果。结对测试一般按照以下步骤实施：

(1) 选择信任的测试人员，该测试人员具有结对测试的理念，结对人员是平等的，不要出现给高级工作人员配备一个初级工作人员的情况。

(2) 选择合适规模的测试项目，不要企图一次测试整个软件，结对测试适合测试一些新的功能，也适合测试那些结对人员都正在参与的项目。

(3) 预先计划好什么时间测试、测试多长时间、中间休息时间等，测试达到的目标、缺陷报告、测试文档、测试用例等，测试的预期目标必须首先定义好，这样参与的测试人员才能有明确的工作目标。

(4) 使用适合两个人在一台机器上工作的环境，要让他们不受干扰，并能自由交流。

(5) 业绩评估，结对测试是否成功？下一次能否做得更好？

习题 5.20

结对测试的优点是什么？

5.21 在线测试(Online Testing)

在线测试，也称动态测试(On-the-fly Testing)是一种测试技术，该技术中测试用例根据一个模型程序来产生，测试执行被结合到一个算法中，该方法在待测试软件运行过程中动态产生测试用例，而不是事先产生测试用例。这种方法的好处如下：

(1) 可以解决在测试反应系统、并行和分布式系统测试中出现的不确定性，避免为了处理各种可能的情况，事先必须产生大量的测试用例。

(2) 离线测试面临的状态空间爆炸问题可以避免，因为在线测试在测试过程中是在大的状态空间中随机抽样，这种做法比穷举各种可能情况要好。

(3) 在线测试可以基于概率分布的动态改变，提供用户选择动作，控制测试场景。

(4) 一次测试可能会持续一个小时、一天甚至更长时间，因此，在线测试可能会执行一些很难、很有压力且时间很长的测试。

(5) 在线测试可以处理一些比较复杂的模型，如非确定性模型，可更好地测试软件系统。

在线测试通过动态变化的策略选择可控制动作,并通过跟踪由于可控制和可观察的动作引起的状态变化,判断这种变化是否符合待测软件的规格说明,如果不符合,则发现故障;否则,存储模型的当前状态信息,产生新的测试动作。在线测试的各种测试动作依赖于各种用户可配置参数,最重要的参数有超时参数和动作权重。超时参数确定在线测试过程中等待被测试程序反应的时间长短;动作权重用来配置在线测试过程中选择某个可控制的动作的策略。

在线测试过程大致可以描述为:运行待测试软件,同时运行在线测试工具,在线测试算法根据捕捉到的软件状态,产生一个测试动作,然后等待被测试软件的反应,如果超时,则发现问题;否则,判断该软件的反应是否正确。如果不正确,则发现错误;如果正确,产生下一个测试动作,继续观察输出,并根据输出及是否超时产生下一步动作,直到发现错误或者测试结束条件满足,停止测试。

在线测试还存在很多问题,如覆盖性度量、测试场景的控制、故障的分析、重现以及避免相同故障的重复出现等问题。

在线测试与自动化测试有点相似,因为在线测试其实就是一种自动化测试技术,但一般的自动化测试,其测试用例和预期输出都是可预知的,而在线测试的测试用例一般都是在测试过程中动态生成的,虽然具有一定的可控制性,但有很大程度的不可预见性。

习题 5.21

为什么要进行在线测试?

5.22 探索性测试(Exploratory Testing)

探索性测试是一种测试方法,把对系统的探索和对系统的测试紧密地结合起来,这种测试方法强调测试人员个人的自由和责任,可以充分发挥他们的创造性和积极性,它把测试过程看成是一种与测试相关的学习,测试设计、测试执行和测试结果的解释同时进行且相互促进的活动,在整个测试项目中,这些活动可以在并行进行的过程中不断优化。因此探索性测试具有以下特点:

(1) 探索性测试强调测试设计和测试执行的同时性,这种同时性是相对传统软件测试过程中“先设计,后执行”来说的。

(2) 测试工程师通过测试来不断学习被测试系统,通过学习来不断改进测试过程。

(3) 探索性测试强调自由性和创造性。

探索性测试并不是一种新的方法,其实每个人都有过探索性测试的经验,除非读者从来没有测试过软件或甚至其他产品。在探索性测试中,测试人员利用在测试中学习得到的经验和获得的关于系统的信息设计更好的测试,整个测试过程是一个不断改进的学习、探索和实践的渐进过程。

探索性测试其实是一门艺术,不同的人会有不同的体验和效果。对测试人员的要求首先是能够系统分析探索软件产品,设计较好的测试用例;其次,能在测试执行过程中认真观察,积极思考,提出很多问题进行下一轮的测试设计;最后,优秀的探索性测试人员应该广泛了解各种测试工具、信息资源、测试数据,甚至广泛结交朋友,在测试过程中,积极使用这

些资源。

习题 5.22

探索性测试的效果是因人而异的，读者认为什么样的人可以将探索性测试的效果做到最好？

5.23 反模型测试(Anti-model Testing)

软件测试通过运行一组精选的测试用例，根据对执行结果的观察，动态验证系统行为，其中测试用例的精心选择可以基于代码(白盒测试或结构化测试)、基于需求(验收测试)、基于规格说明(黑盒测试)以及设计模型(基于模型的测试)等。

基于模型的软件测试是通过描述软件的模型来获得测试用例，这个模型可以来自规格说明或者是由软件工程师用图形工具设计出来的。一般来说，测试用例可以由这些工具自动生成。通过执行这些自动生成的测试用例，可以确认实现的系统是否满足规格说明的要求。尽管基于模型的软件测试非常有用，也非常有效，但在很多情况下不太好用，原因有 3 个：一是形式化模型的建立和使用对人们的要求比较高，一般需要人们精通形式化方法和工具，形式化理论、方法和工具往往都比较复杂；二是模型一般都是对系统一定程度的抽象，由模型产生的测试用例一般不能直接测试，需要进行进一步处理，使其更加具体；三是在某些领域，如一些遗产系统或组件系统，没有现成的关于软件的形式化模型可用。

反模型测试方法与基于模型的测试方法相反，它一开始没有模型可用，通过执行一些抽样出来的测试用例，观察系统行为，并将这些信息综合起来，通过分析推理形成关于系统的一个抽象模型。无论基于模型的测试还是反模型测试，最后都会有一个关于系统的模型在一边，另一边是系统的实现，然后通过测试执行，确定系统实现与系统模型是否一致。

图 5-17 给出了反模型测试方法的基本步骤如下：

(1) 是在待测试软件的模型无法知道的情况下，要根据待测试软件的使用剖面等可以获得的信息，抽样选取一些适当的测试用例，并给出预期输出。

(2) 执行测试用例，收集执行轨迹信息，可以通过一些监控程序完成信息的收集工作。

(3) 分析收集到的执行信息，合成系统模型，如果已经有一个初步模型，需要核实确认这个初步模型的正确性，如果存在问题，需要修正这个模型。

(4) 根据得到的模型进一步选择测试用例，转第(2)步，继续执行测试用例，收集执行轨迹信息，到第(3)步，确认模型正确性、修正模型等，直到测试资源耗尽或者测试人员满意测试结果，测试结束。

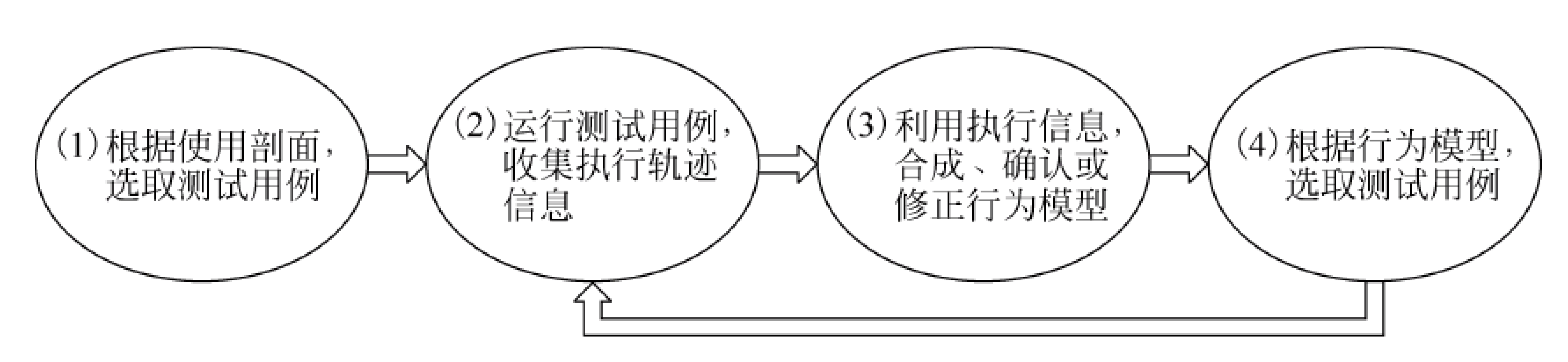


图 5-17 反模型测试方法的步骤

习题 5.23

反模型测试与基于模型的测试区别是什么?

5.24 成分测试(Compositional Testing)

软件规模和复杂性的增加已经成为软件测试研究和发展的重要挑战,人们利用解决复杂问题的传统方法——分治法,把复杂的大规模软件测试分解为测试该软件的各个组成成分,其实基于组件的软件开发就是把开发复杂的大规模软件开发分而治之,专心开发各个组成部件,然后把这些部件集成起来,组成要开发的软件系统。人们已经提出很多技术和工具来递增地测试这些软件组成成分。例如,人们提出很多测试顺序及测试策略,以达到最少使用驱动模块和桩模块。成分测试的一个重要课题就是,当软件每个部件已经得到充分测试之后,如何在集成测试过程重用这些信息,如何利用每个成分的质量信息,结合系统集成测试信息,推理整个系统的质量水平。基于组件的软件测试就是一种成分测试。到目前为止,人们对于各个组成成分的正确性与整个系统的正确性的关系,以及在各个组件充分测试之后,还要进行多大程度的集成测试、怎样设计集成测试等问题还不是十分清楚。

习题 5.24

为什么要进行成分测试?

5.25 有限状态机测试(FSM Testing)

有限状态机(Finite State Machine, FSM)作为一种形式建模语言已被广泛用于描述软件的需求,基于 FSM 的测试方法亦被广泛研究。

由 FSM 产生测试用例的思想来源于有向图的遍历。基于图形覆盖的程度,研究人员提出了状态覆盖和迁移覆盖两类基本覆盖,并给出了由状态覆盖和迁移覆盖生成测试用例的方法。依据有向图的遍历方法,由初态开始遍历 FSM,生成一棵广度优先生成树,再获得由树根结点到叶子结点的迁移序列作为测试用例。迁移之间可能存在一些约束关系,例如,BBS 论坛中存在两类访客,一类是未注册的游客,另一类为注册用户。注册用户可以发表帖子和浏览其他用户的帖子,而游客只能浏览已发表的帖子,因此注册用户与发表帖子之间存在约束关系。

软件测试中可能存在一类额外迁移错误,即描述需求的规格说明中并没有这些迁移,而实际应用中却存在这些额外的迁移。在某些情况下,这些额外的迁移会导致系统的崩溃。例如 1999 年 12 月 3 日,美国航天局火星极地登录者号探测器飞船在试图登录火星表面时失踪。错误修正委员会观测到故障,认定出现失误动作的原因极有可能是某一个数据被意外更改。这表明该系统中至少存在一条额外迁移,在正常情况下,这条额外迁移并不会被执行,但在特定情况下,这条额外迁移的执行会导致系统运行出错。前面讨论的基于 FSM 的测试方法均不能探测到这类错误,因为上述方法都是由规格说明导出测试用例,而额外迁移仅仅出现在实际应用中,因此,由规格说明产生的测试序列并不能激活实际应用

中的额外迁移。虽然额外迁移的错误已经超出了基于 FSM 的测试能力，但仍然存在少量基于 FSM 的测试方法能够探测这类错误。例如随机测试生成方法，该方法随机产生测试输入，直到 FSM 中的所有迁移都被至少覆盖一次。传统随机测试生成方法直接由规格说明随机产出覆盖 FSM 的测试序列，因此同样也不能发现额外迁移错误。

习题 5.25

什么是 FSM？查阅 FSM 方面的资料，更全面地了解 FSM 测试的理论、方法和应用。

5.26 基于 Petri 网的测试(Petri Net Based Testing)

Petri 网是一种图形化的形式化语言表示法，该方法采用具有形式语义的图形语言，既有严格的数学定义，又有直观的图形表示。图形化表示法易于理解，而且非专业人员能够使用，严格的数学定义使其具备严密性、逻辑性和精确性，因此是一种通用的系统确定表示法，适合于描述异步并发现象的系统模型。

作为建模工具的一种，Petri 网是 1962 年由 Petri C. A 在他的博士论文中作为网状结构的信息流模型提出来的。近年来，Petri 网以及它的各种改进模型如有色 Petri 网等被广泛地应用到了众多复杂系统的离散事件仿真模型搭建工作中，为离散事件的仿真提供了不少新的方法。相对于传统的马尔可夫链等建模方法，Petri 网作为一种形式化方法，它能分析系统软硬件中多方面的特性，如功能性、安全性、可靠性、实时性等。

如何系统有效地进行基于 Petri 网的软件测试一直是人们研究的热点，这个问题包括包括 3 个方面：如何定义基于 Petri 网的可以客观度量的测试标准？怎样根据 Petri 网自动有效地生成测试用例？如何有效地控制整个测试过程？

一方面，由于 Petri 网作为软件系统的描述模型，它可以作为待测试软件的一种规格说明，因此可以基于 Petri 网的软件测试可以作为一种基于规格说明的软件测试，测试时，根据 Petri 网提供的需求描述信息来设计测试用例，进行基于规格说明的测试或者黑盒测试。另一方面，Petri 网是一种可以执行的模型，需要根据其他规格说明设计测试用例，验证该 Petri 网模型是否正确，或者根据 Petri 网模型设计测试用例验证是否与其他规格说明一致，进行所谓的白盒测试(基于程序的测试)。

目前关于基于 Petri 网的软件测试，人们提出了面向状态的测试、面向迁移的测试、面向数据流的测试和面向规格说明的测试 4 种方法。这几种方法都是传统的黑盒测试和白盒测试在把 Petri 网模型分别作为软件规格说明和可执行程序时的应用。更多内容可以参考第 2 章中的各节，例如，第 2.1.4 小节中数据流测试，第 2.2.5 小节中状态转换测试等。

基于 Petri 网的软件测试与基于 FSM 的软件测试有着很大的相似性，这两种模型有着根本的不同，例如，FSM 一般是顺序性的，它定义的状态一般都是全局的状态；而 Petri 网一般是并行的，它定义的状态一般都是分布式的。

习题 5.26

什么是 Petri 网？查阅 Petri 网方面的资料，更全面地了解 Petri 网的理论、方法和应用。

5.27　基于模型检查的测试(Model Checking Based Testing)

模型检测的研究始于21世纪80年代初，当时Clarke、Emerson等人提出了用于描述并发系统性质的CTL(Computation Tree Logic，计算树逻辑)逻辑，设计了检测有穷状态系统是否满足给定CTL公式的算法，并实现了一个原型系统。这一工作为对并发系统的性质自动进行验证开辟了一条新的途径，成为近20年来计算机科学基础研究的一个热点。随后不久出现的符号模型检测技术使这一方法向实际应用性迈出了关键的一步。

模型检测的基本思想是用状态迁移系统(S)表示系统的行为，用模态/时序逻辑公式(F)描述系统的性质。这样“系统是否具有所期望的性质?”就转化为数学问题“状态迁移系统S是否是公式F的一个模型?”，用公式表示为“S F?”。对有穷状态系统，这个问题是可判定的，即可以用计算机程序在有限时间内自动确定。

早期的模型检测侧重于硬件设计的验证，随着研究的进展，模型检测的应用范围逐步扩大，涵盖了通信协议、安全协议、控制系统和一部分软件。电子线路设计验证的例子包括先进先出存储器的验证，验证的性质包括输入和输出的关系。浮点运算部件的验证，验证的性质包括计算过程所需满足的不变式。

对于复杂的协议和软件，模型检测的验证基于抽象和简化的模型，从验证角度来讲，这并非十分严谨。更为确切的说法应该是协议和软件的分析。协议验证的例子包括认证协议的验证，验证的性质包括对通话双方的确认；合同协议的验证，验证的性质包括公平和滥用的可能性；缓存协议的验证，验证的性质包括数据的一致性和读写的活性。对于软件，由于软件的复杂和软件运行中可能到达的状态个数通常没有实质上的限制，验证通常局限于软件的某些重要组成部分。软件验证的例子包括飞行系统软件的验证，验证的性质包括系统所处的状态和可执行动作之间的关系；铁路信号系统软件的验证，验证的性质包括控制信号与控制装置状态的关系。模型检测还可以应用于其他方面，其基本思想是将一个过程或系统抽象成一个有穷状态模型，加以分析验证。这方面的例子包括化学过程验证，验证的性质包括阀门、管道和容器的状况；公司操作过程分析，分析的内容包括操作过程是否具有所需的功能；电站操作程序的验证，验证的性质包括操作程序的动作前后关系和操作是否安全可靠。

测试是保证软件产品可靠性和正确性的传统手段，与模型检测不同，测试是对选定的输入数据集运行系统，通过比较所产生的输出与预期值来判断系统是否会有错误。测试的主要局限性在于：对给定数据集通过了测试并不能保证在实际运行中对其他输入不发生错误。

与测试不同，模型检测不是针对某组输入，而是面向某类性质来检查系统是否合乎规约。在系统不满足所要求的性质时，模型检测算法会产生一个反例(一般是一条执行路径)来说明不满足的原因。这一功能与测试有异曲同工之处。如何将模型检测与测试技术相结合，使两者相辅相成，是一个十分值得注意的研究方向。

模型检测在硬件和通讯协议的分析与验证中已经取得了很大的成功，如何将这一技术应用于软件，是目前非常活跃的研究领域。软件由于涉及无穷数据域上的运算而往往呈现出无穷状态，这是软件模型检测的主要难点，但是很多系统(如控制软件)并不涉及复杂的数

值计算，往往只用到线性实数约束或整数加减等简单的运算，因此可以采用诸如线性约束求解等技术实现模型检测，另外也可应用抽象等技术映射到有穷域上进行模型检测。

习题 5.27

查阅模型检测方面的资料，了解模型检测的理论、方法和应用。

5.28 TTCN 测试（TTCN Testing）

TTCN（Testing and Test Control Notation）是一种用于测试的编程语言，特别是用于通信协议测试和 Web 服务测试。TTCN 测试用例集是一组利用 TTCN 编程语言编写的测试用例组成的集合。TTCN-1 和 TTCN-2 都是利用树和表来表示的（此时 TTCN 是 Tree and Table Combined Notation 的缩写），TTCN-3 比较灵活，可以用于一般的软件测试，而不仅仅是协议测试。

TTCN 是由 ISO/IEC 9646 和 ITUX.292 系列所提出的实现 OSI 与 ITU 协议定义的一致性测试方法的标准。由于 TTCN 集成开发环境 ITEX（Interactive TTCN Editor and eXecutor）能够自动生成 ANSI C 的测试代码，因此测试的主要工作量集中在利用形式化语言描述测试流程上，这同时避免了由于手工编程可能导致的错误。测试过程出现的问题可由 ITEX 生成相应的测试报告，减少跟踪调试程序的工作量。并且所有与具体平台相关的代码都被集中在测试的内核中。因此，TTCN 所生成的测试模块可重用性好、容易维护。TTCN 与 ASN.1（Abstract Syntax Notation.1）兼容，ASN.1 属于国际标准 ISO/IEC 8824 系列，是开放系统互联（OSI）的关键组成部分之一，由于两者的兼容性，使 TTCN 的应用更加广泛。

然而，最初的 TTCN 不能设计和描述并行行为，人们很快意识到对 TTCN 的并行能力的扩展的重要性和迫切性。这也是 TTCN-2 出现的直接原因。尽管在 TTCN-2 中做了扩展，但是对于新出现的不同领域的不同种类的测试仍存在很多缺陷和不足，STF（Special Task Force）133 等从 1998 年开始研究新版 TTCN，到 2000 年 10 月完成，新版的 TTCN 命名为 Testing and Test Control Notation version 3（TTCN-3）。

TTCN-3 最顶层单元是模块，它内部不能再有子模块。TTCN-3 模块之间相互独立，它们可以通过 Import 语义共享数据定义。一个测试套就是一个模块。一个模块有两部分：定义部分和控制部分。定义部分定义了测试组件、通讯端口、数据类型、常量、测试数据模板、函数、端口程序呼叫信号、测试用例等。控制部分包含局部变量定义、调用测试用例并控制其执行顺序。

TTCN-3 规范对 TTCN-3 测试系统的概念模型进行了描述。TTCN-3 测试系统由一组具有特定功能的实体组成。这些实体管理测试的顺序，解释和执行已经编译过的 TTCN-3 代码，实现和被测系统的正确通信以及实现外部函数（在 TTCN-3 模块外面定义，在模块中申明为外部函数）和处理定时器的操作等。TTCN-3 测试系统分解为测试管理（Test Management，TM）、测试执行实体（TTCN-3 Executable，TE）、SUT 适配器（System Under Test Adapter，SA）和测试平台适配器（Platform Adaptor，PA）。

TTCN-3 测试系统中的两个主要的接口：TTCN-3 控制接口（TTCN-3 Control Interface，TCI）和 TTCN-3 运行时接口（TTCN-3 Runtime Interface，TRI）。它们分别制定了 TM 和 TE 之间的接口，以及 TE 与适配器（SA 和 PA）之间的接口。目前，TTCN-3 规范只对 TRI 进行了接口定义，而给予测试工具提供商在测试系统的实现中很大的灵活性。而一般情况下，TRI 需要由 Java 或 C/C++等语言来开发。

TTCN-3 可以用作多种通信端口上的各种响应系统测试的描述语言。典型的应用领域是协议测试（包括移动协议和互联网协议）、服务测试（包括增补服务）、模块测试等。TTCN-3 并不仅仅局限于一致性测试，它可用于多种类型的测试，如互操作性测试（Interoperability Testing）、性能测试（Performance Testing）、鲁棒性测试（Robustness Testing）、回归测试（Regression Testing）、系统和集成测试（System and Integration Testing）。

关于 TTCN 的更多内容可以查阅 http://www.ttcn-3.net，或者维基百科等。

习题 5.28

查阅 TTCN 方面的资料，了解 TTCN 的理论、方法和应用。

5.29 布尔规格测试（Boolean Specification Testing）

在软件规格说明中，布尔表达式一直是人们关注的焦点，因为布尔表达式非常容易引入错误。在航空、医疗以及一些控制软件等重要的安全关键软件中，布尔表达式可以用作描述规格的条件、模型谓词以及逻辑表达式等，如何根据这些布尔表达式设计测试用例来测试根据这些布尔规格说明建立起来的软件，并发现其中隐藏的错误是布尔规格测试的关键问题。

处于程序或者规格说明中的布尔谓词 p，如果它包含 n 个布尔变量，需要验证 p 的正确性的测试用例就需要 2^n 条，一般情况下，布尔变量的个数 n 是比较大的，例如，在一个电子航班系统中，就有 30 多个条件变量。因此对布尔规格进行遍历测试一般是不可行的，必须要采用一些科学有效的方法进行低成本的测试。常用的测试方法包括因果图法（详见第2.2.3小节）、修改决策条件方法（MC/DC，详见第 2.1.7 小节）。一个非常重要的方法是基于故障类型的测试方法，表 5-6 列出了各种错误类型的定义和实例。

表 5-6　布尔表达式的错误类型及实例

错误类型	解　释	以 $(a+b)(\neg a+c)$ 为例子
Operator Reference Fault (ORF)	运算符＋被·错误代替，或相反	$(ab)(\neg a+c)$
Expression Negation Fault (ENF)	表达式被错误取反	$\neg(a+b)(\neg a+c)$
Variable Negation Fault (VNF)	原子变量被错误取反	$(\neg a+b)(\neg a+c)$
Associative Shift Fault (ASF)	结合错误，如括号范围实现错误	$(a+(b\neg a)+c)$
Missing Variable Fault (MVF)	表达式中变量丢失	$b(\neg a+c)$
Variable Reference Fault (VRF)	表达式中变量引用错误	$(a+a)(\neg a+c)$
Clause Conjunction Fault (CCF)	变量 a 被 $a\cdot b$ 代替	$(ab+b)(\neg a+c)$
Clause Disjunction Fault (CDF)	变量 a 被 $a+c$ 代替	$(a+c+b)(\neg a+c)$
Stuck at 0(SA0)	变量 a 被 0 代替	0
Stuck at 1(SA1)	变量 a 被 1 代替	bc

这些错误类型之间具有如图 5-18 所示的相互包含关系，在测试过程中可以根据具体的测试要求选择合适的测试标准进行测试。

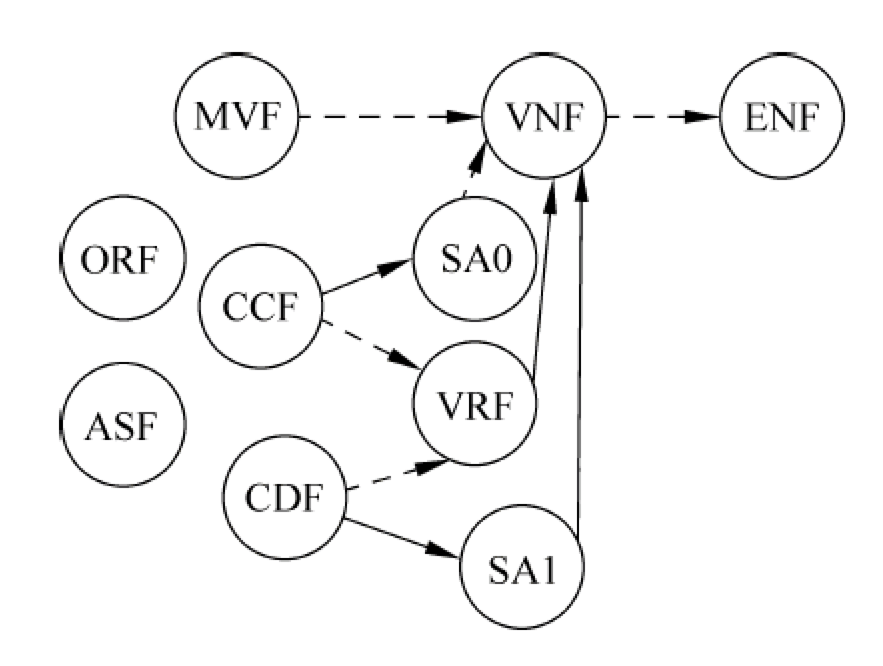


图 5-18 错误类型之间的相互关系

习题 5.29

查阅相关参考文献，系统了解布尔规格说明测试的有关研究。

5.30 基于统一建模语言测试（UML Based Testing）

UML（Unified Modeling Language）是一种可视化的面向对象系统建模描述工具，它继承了 Booch、Jacobson 和 Rumbaugh 等人提出相关概念和方法，于 1997 年被 OMG（Object Management Group）组织采纳为面向对象系统的建模标准。UML 为软件需求分析、设计等不同阶段提供了各种图形工具，可以描述和表达各种用户需求和系统设计。

随着面向对象方法日益得到重视和应用，人们提出了面向对象软件的测试方法，而 UML 是常用的面向对象系统建模工具，因此，有必要建立基于 UML 的测试方法。所谓基于 UML 的测试，就是应用待测试软件系统的 UML 模型来获得软件测试的需求和覆盖准则。UML 中使用以下图形：

（1）用例图（User Case Diagram）。描述软件系统的功能和使用者。

（2）类图（Class Diagram）。描述软件系统中类的静态结构。

（3）对象图（Object Diagram）。描述软件系统在某个时刻的静态结构。

（4）包图（Package Diagram）。把设计元素进行分组的一种通用组织机制。

（5）序列图（Sequence Diagram）。按时间顺序描述软件系统对象之间，软件系统与外部环境之间的交互。

（6）协作图（Collaboration Diagram）。按时空顺序描述软件系统对象间的交互和协作关系，可与序列图相互转换，统称交互图。

（7）状态图（Statechart Diagram）。描述系统对象状态及其迁移关系。

（8）活动图（Activity Diagram）。描述软件系统中各种活动的流程。

（9）构件图（Component Diagram）。描述了软件系统实现的物理结构。

（10）部署图（Deployment Diagram）。描述了软件系统在硬件环境，特别是网络或分布式环境中的配置关系。

针对以上各种UML图形，人们提出了相应的测试充分性准则，例如，类图测试充分性准则包括3种：关联乘法准则要求每个与该类关联的类组合对都要被测试到；类泛化准则要求每个类的具体实现都要被测试到；类属性准则要求每个类属性都要被测试到。用例图准则要求每个用例步骤、用例场景或者用例路径都需要被覆盖等不同标准。实际上也是把UML分别作为规格说明和程序本身进行黑盒和白盒测试而提出各种相应的测试标准，这里不再赘述。

习题 5.30

什么是UML？查阅UML方面的资料，更全面地了解UML和它的应用。

第6章 特定应用软件(X-软件)的测试

为了提高软件开发的效率、降低成本和保证软件质量，人们提出了很多新的开发方法，利用这些方法开发的软件，如面向对象软件、面向方面软件、面向服务软件和构件软件等，与传统的软件既保持着很大的相似性，又有着非常明显的特殊性。同时，随着软件应用领域的扩展，特别是网络技术的发展，产生了一批适用不同应用场景的软件，例如，嵌入式软件、Web 应用软件、云计算环境下的软件、物联网环境下的软件、普适计算环境下的软件、多核并行软件等，特别是，近几年来人们提出了高可信软件的概念，给软件测试提出了很多新的挑战。

无论使用什么新方法开发的软件，无论是运行在什么样平台上的软件，它们都要在开发的不同阶段(适用第 3 章开发过程中的测试)，针对软件不同方面(适用第 4 章软件特性及各方面的测试)，利用不同测试技术(适用第 5 章特殊的软件测试技术)进行黑盒测试和白盒测试(适用第 2 章白盒测试和黑盒测试)。

本章将分别介绍这些面向新开发方法以及新运行平台的 X-软件进行 Y-测试的方法，针对这些 X-软件的 Y-测试方法，已经有一些专门的著作介绍，因此本书只注重介绍 X-软件的特殊性，而对 Y-测试很多共性的内容将不再赘述。

这里对 X-软件进行 Y-测试可以进行如下具体表述：

X-软件　X＝{面向对象，面向服务，面向方面，构件，普适计算，Web，云计算，物联网，嵌入式，并行，高可信，网构}，本章将逐一介绍。

Y-测试　在不同开发阶段可以表示为 Y＝{单元，集成，系统，回归，验收，α，β，γ}，详见第 3 章。

针对 X-软件的不同方面进行的 Y-测试，Y＝{负载，压力，性能，可靠性，容量，安装，可用性，稳定性，本地化和国际化，可访问性，授权，一致性，配置，文档，兼容性，接口，可恢复性，健壮性，协议，在线帮助，能力，卸载，备份，数据转换，人机界面，Playtest，容错，余量，穿越}，详见第 4 章。

对 X-软件使用不同 Y-测试技术时，Y＝{组合，蜕变，基于规格说明的，基于模型的，基于错误的，基于搜索的，统计，基于操作剖面的，变异，基于性质的，极限，模糊，适应性，图形用户界面，随机，在线，探索性，成分，有限状态机，Petri 网，TTCN，UML，Concolic，结对，反模型，反随机，自适应随机，布尔规格，基于模型检验的}，详见第 5 章。

同时，Y-测试也可以宏观地表示为黑盒测试和白盒测试，Y＝{黑盒，白盒}，而黑盒测试包括等价类划分、边界值分析、因果图和决策表、错误猜测、状态转换、语法测试等 6 项技术，白盒测试包括语句覆盖、路径覆盖等 10 项不同程度的白盒测试技术，详细见第 2 章。

6.1 面向对象软件的测试(Object Oriented Software Testing)

面向对象技术在软件工程中的推广使用,使得传统的测试技术和方法受到了极大的冲击,对面向对象技术所引入的新特点,传统的测试技术已经无法有效地对面向对象软件进行测试,因此,必须针对面向对象程序的特点,研究相应的测试方法和测试策略。

1. 面向对象软件基础

面向对象可以看成是现实世界模型的自然延伸,现实世界中的任何实体都可以看成是对象。传统的过程式编程语言是以过程为中心,以算法为驱动,在过程式语言中,程序=算法+数据;面向对象的编程语言是以对象为中心,以消息为驱动,在面向对象的语言中,程序=对象+消息。面向对象的软件开发以对象、类、继承、多态、消息和接口6个重要的基本概念为核心。

(1) 对象(Object)是一个可操作的实体,既包含了特定的数据(属性),也包含操作这些数据的代码(方法),对象是一个独立的基本的可计算实体,程序由很多对象组成,对象之间通过消息相互作用。在测试过程中,对象是测试的重要目标,例如,在程序运行时,对象的行为是否符合规格说明?对象是否能够与相关对象进行协同工作?针对对象的测试要考虑的内容包括:

- 对象的封装 通过封装将数据与操作结合起来,使得程序更加紧凑,简化了对对象的使用,但封装导致的信息隐藏使得针对信息或数据的测试更加困难。
- 对象的状态 同一个类会有多个不同的对象,每个对象都有自己的状态,在测试过程中,应测试到对象的每个状态,并观察对象能否处于所有不同的状态,对象的状态转换是否正确。
- 对象的生命周期 每个对象都经过创建、访问、修改、删除的过程,在这个生命周期中,应从多方面测试对象的状态是否与生命周期相符合。
- 对象的交互 不同对象之间通过消息的发送和接收产生关系,应测试到各种可能的交互情况。

(2) 类(Class)是通过抽象数据类型的方法实现的一种数据类型,是具有共性的对象集合。类是对某一类型对象的抽象,而对象是某一种类的实例。创建对象的过程称为实例化,创建的结果称为实例。类的声明描述了类代表什么,以及相应的对象能做什么,包括了多个属性和方法。对于类的实现需要关注如下内容:

- 构造函数 构造函数用于创建新的对象,并对其进行初始化。首先要注意测试各种不同的对象创建方式,检查初始化过程是否正确,是否存在遗漏的或错误初始化的属性。其次,若存在有属性是其他类的对象的情况,则应检查对其他类的实现是否正确。
- 析构函数 析构函数用于对删除的对象进行处理,需要检查所有申请的内部数据空间是否得到及时释放、重要属性是否恢复正常的取值。
- 方法 类通过定义各种方法实现对外接口,使得对象内部的数据可以与外界进行交换,以实现对象之间的交互。需要对类中各种方法进行检查。同时,还要检查的类

的实现是否满足相关的设计原则。

封装(Encapsulation)是将数据以及操作数据的方法关联起来,构成一个类,将其内部的实现细节隐藏在这个类中,外界无法访问,使得相应对象封闭保护起来。封装简化了对对象的使用,人们只要知道对象的输入输出,无须了解内部细节。封装使得对象内部数据的访问只能通过接口来进行,从外部无法了解内部数据的状态,这给测试带来一定的难度。封装要求对象应具有明确的功能,并留有若干接口以便于和其他对象之间进行相互作用,因此,类应该具有高内聚度和低耦合度。

面向对象软件的基本单位是类,一个类封装了多个属性和操作,类的操作通常需要依赖于属性,操作的许多功能需要在不同的实例状态下才能展示出来,甚至有的操作需要在特定的实例状态下才能正常执行。因此,在测试面向对象软件时,不能简单地对每个类的各个操作进行测试,在调用任何一个操作之前必须保证相应的实例处于该操作需要的预期状态下。换一个角度,一个测试用例不仅仅调用一个操作,实际上是对整个类进行测试的一个调用序列,因此在设计每个类的测试用例时,不仅仅要考虑调用各个操作,还要考虑如何设计调用序列。

(3) 继承(Inheritance)是类之间的一种关系,通过从某个类(父类)派生可以得到一个新的类(子类或派生类),该类不仅具有父类的特点和功能,实现对父类的复用,同时还可以对父类进行扩展,具有新的特点和功能,不需要和父类保持相同或一致规格。继承可以实现将一个通用对象中的特殊性不断提炼出来,得到逐渐特殊化的类,从而建立对现实世界的分层次的描述。继承对于增加软件的可扩充性、提高代码的重用度是非常重要的。然而继承机制给测试带来两个重要问题：其一是缺陷传播,继承机制给父类缺陷向子类传播提供了途径；其二是重复测试,因为子类从父类继承下来的方法往往需要重新测试。

面向对象软件中的继承机制使得传统的测试充分性准则,如语句覆盖、分支覆盖等不再完全适用。例如,类 B 继承 A,若 A 有两个属性 u 和 v 以及三个操作 a、b 和 c,类 B 除了继承这些属性和操作之外,又增加了属性 x 和 y 以及两个操作 d 和 e。如果已经对 A 进行了充分的测试,那么在测试 B 时只对 B 自身定义的属性和操作进行测试是不充分的,因为类 B 在语法上有 4 个属性和 5 个操作,对 B 的充分测试应该重新考虑这些属性和操作。

继承的使用可能造成许多类的代码难以理解,例如,在很深的继承上处于叶结点的类,可能本身的代码很少,但却可以通过继承拥有丰富的特征,这样很可能导致使用时发生误用。例如,类实例初始化就是一个经常出现误用的地方,在 C++ 中每个类都可以定义若干构造函数,用于实例的初始化,在子类的构造函数中通常只初始化该类本身定义的属性,并通过调用父类的构造函数来初始化其在祖先类中定义的属性。在选择使用哪个构造函数来初始化类的实例时,程序员常常可能会忽略该构造函数是否会最终正确地初始化定义在其各个祖先类中的属性。

(4) 多态(Polymorphism)即所谓一个对外接口,多种内在实现方法,特点是同样的方法可以应用于不同对象,根据特定的对象类型与某个实现绑定。绑定可以分为静态绑定和动态绑定。静态绑定是指在编译时刻就完成的绑定,而动态绑定是指在运行时刻完成的绑定。多态可以分为参数多态和包含多态,前者是根据多个参数来定义一种类型,后者则是同一个类具有不同表现形式,并可通过继承或接口来实现。对于多态的测试,一般需要通过动态测试来检查,静态测试无法保证测试的正确性。因为一个函数名可以绑定多个实例,而具体绑

定哪个实现需要在运行时才能决定,这样一个函数名起到了 Switch 语句的作用,根据不同的输入调用不同的实现。因此从语句覆盖的角度看,仅仅覆盖了该函数名所在的语句并不一定覆盖整个 Switch 语句,只有覆盖了该函数名调用各种实现的情况才能覆盖该 Switch 语句。

在面向对象的软件中,继承和多态结合在一起可以产生多种变化,这一方面可以帮助程序员设计出许多精巧的代码,也使得因使用不当而引起的错误难以被测试发现。

(5) 消息(Message),对象间通过传递消息进行相互作用,通过消息来请求执行某个操作,消息的测试需要考虑消息的发送者何时可以发送消息、消息的接收者如何接收消息、如何处理非预期的消息以及消息所包含的参数是什么、是否可以修改、修改是否正确等。

(6) 接口(Interface)用于描述类对象的一系列规范的行为,接口的测试要检查接口包含的行为与对应的类的行为是否一致,接口往往与其他接口或类具有相互作用,应测试所有可能的相互作用。

在面向对象程序的单元测试过程中,由于一个类的各个操作通常是相互依赖的,特别是由于继承的存在,一个类通常依赖于其父类或其他祖先类。因此通常很难对一个类中的单个操作进行充分的单元测试。

2. 面向对象软件测试的不同层次及特点

面向对象软件测试可分为 4 个层次:方法测试、类测试、类簇测试和系统测试,其中方法测试也叫行为测试,是指对类中的各个方法(定义的各种操作)进行单独的测试;类测试的重点是类内方法的交互和其对象的各个状态;类簇也叫子系统,由若干个类组成,类簇测试相当于集成测试,重点测试一组协同操作类之间的相互作用;系统测试检验所有类和主程序构成的整个软件系统是否符合要求。

由于方法测试不能独立于类,一般看成为类测试的一部分,因此面向对象软件测试一般分为 3 个层次,其中面向对象单元测试对类的成员函数及成员函数间的交互进行测试,面向对象的单元测试是面向对象的集成测试的基础,面向对象集成测试对系统内部的相互服务进行测试,如成员函数间的相互作用、类间的消息传递等。面向对象系统测试是基于集成测试的最后阶段的测试,主要以用户需求为测试标准,检验整个软件系统是否符合需求。

类是面向对象软件组成和运行的基本单元,面向对象软件的内部实际上是各个类之间的相互作用,对它的测试也就显得更加非常重要。在面向对象系统中,系统的基本构造是封装了数据和方法的类和对象,而不是一个个能完成特定功能的功能模块。每个对象都有自己的生存周期,有自己的状态。消息是对象之间相互请求和协作的途径,是外界使用对象方法及获取对象状态的唯一方式。对象的功能是在消息的触发下,由对象所属类中定义的方法与相关对象的合作共同完成,且在不同状态下对消息的响应可能完全不同。所以对类进行测试时必须考虑类的对象所具有的状态,着重考察一个对象接收到一个消息序列之后,是否达到了一个正确的状态。因此,类测试的重点是类内方法间的交互和其对象的各个状态,类的测试用例主要由方法序列集和相应的成员变量的取值组成。

一般单元测试的理论和方法都是可以应用到类测试的,例如,类测试的内容、测试时间、测试人员、测试方法、测试用例设计方法和测试充分性等,与单元测试是一样的。同样集成

测试、系统测试的基本理论和方法也都可以直接应用到面向对象软件的测试。

为了能够尽早地发现面向对象软件中可能存在的错误，与一般的软件测试相似，人们建立了图 6-1 中相应的面向对象软件测试模型。

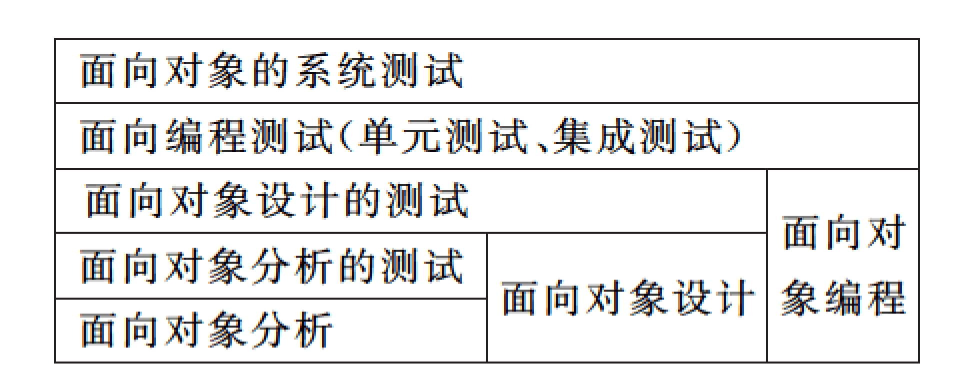


图 6-1 面向对象软件测试模型

（1）面向对象分析主要检查分析结果是否符合相应的面向对象分析方法的要求，是否可以满足软件需求。目前已存在多种面向对象分析方法，例如，对于 UML 用例模型，首先确定每个用例中的参与者，根据他们的职责审查各个用例和它们的规格说明，检查在功能上是否已经覆盖了所有需求以及其他不符合之处。对于 UML 结构模型，针对类图中的类、关系、包、属性、操作等，检查在数据对象上不符合之处，还需要利用走查等方法，审查 UML 的由活动图、状态图、顺序图等组成的行为模型，检查在系统服务方面的不符合之处，以及是否能够保证各个用例的顺利完成。

（2）面向对象设计的测试与面向对象分析的测试没有明显的界限，因为设计结果都是分析结果的扩展和延伸，针对设计的测试除了对设计结果的测试之外，还要检查设计结果与分析结果的一致性，设计结果对编程的支持。

（3）面向对象编程的测试一方面需要执行代码进行测试，另一方面还要检查程序代码的风格。对于一定规模的程序，将整个程序放在一起进行充分的测试比较困难，一般都是将程序的各个部分分别进行测试，同时由于程序的不同组成部分之间存在依赖关系，在测试一个需要依赖其他部分时，需要把其他部分集成在一起进行集成测试，此时测试的重点也主要是测试各个部分之间的交互。在面向对象软件测试中，人们一般把可单独执行的实体的测试称为单元测试，而把多个实体单元集成在一起的测试称为集成测试；在软件交付使用前还需要对整个软件系统进行全面的测试，确认达到用户需求，称之为系统测试。

① 面向对象的单元测试。传统的单元测试是针对程序的函数、过程或完成某一特定功能的程序所进行的测试，面向对象的单元测试是针对面向对象程序的基本单元——类，为此，分两边走，测试与类相关的操作和测试类。

操作级的测试主要关注那些功能单一、调用频繁的方法(操作)可能出现的不易发现的错误，例如，操作 strchr()用于查找最后的匹配字符，但程序中误写成了查找第一个匹配字符。将 if (strncmp(str1, str2, strlen(str2))) 误写成 if (strncmp (str1, str2, strlen (str1)))，此时如果使用的数据 str1 和 str2 长度相同，就无法检测错误。因此在设计测试用例时，应对函数返回值作为条件判断等情形以及字符串操作等情况特别注意。

面向对象编程的特性使得成员函数的测试不同于传统的函数或过程测试，例如，继承的操作可能需要重新测试，特别是在子类中继承过来的成员函数做了改动，或者成员函数调用了改动过的成员函数。值得注意的是，父类的测试用例不能照搬到子类。例如，基类中 Base ::Redefined()含有如下语句 if(value＜0)message("less")； else if (value＝＝0) message

("equal"); else message("more"); 在子类中 Derived:: Redefined 中定义为 if(value<0) message("less"); else if (value==0) message("it is equal"); else {message("more"); if (value==88) message("luck")};在原有的测试上,对 Derived:: Redefined 的测试只要做如下改动:改动 value==0 的预期输出,增加 value==88 的测试。

对于类的测试需要覆盖类中所有操作、类中所有属性的设置和访问、类的对象的所有可能的状态转换,所有状态转换的事件都要模拟到。

② 面向对象的集成测试。面向对象的集成测试又叫交互测试,目的是确保对象的消息传递能够正确进行,可用 3 种方法:用例或基于场景的测试、基于线索的测试和对象交互测试。其中用例或场景描述了系统的使用模式,测试可以根据场景描述和对象簇制定,这种测试侧重于系统结构,首先测试那些几乎不使用服务器的独立类,再测试那些使用了独立类的下一层次的(依赖)类,这样一层一层地持续下去,直到整个系统构造完成。基于线索的测试把响应某一系统输入或事件所需的一组类的实例集成在一起,形成一条线索。每条线索将分别集成和测试,因为面向对象系统通常是事件驱动的,所以这是一个特别适合的测试形式。对象交互测试提出了集成测试的中间层概念,中间层给出叫做"方法-消息"路径的对象交互序列,输入事件+"方法-消息"+输出事件="原子"系统功能。

集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误,关注的是系统结构和内部的相互作用,面向对象的集成测试也包括静态测试和动态测试。静态测试对程序的结构进行检查,检查是否符合设计要求,是否存在结构和实现上的缺陷,是否符合设计要求。动态测试在设计测试用例时,参考功能调用结构图、类关系图或实体关系图等确定不需要重复测试的部分,从而优化测试用例,减少测试工作量,使得进行的测试能够达到一定覆盖标准,例如,达到类所有的服务要求或服务提供的一定覆盖率;达到对所有消息驱动的线程的一定的覆盖率;达到类的所有状态的一定覆盖率;以及使用现有测试工具得到一定的代码执行覆盖率。

习题 6.1

什么是面向对象的计算范型?查阅面向对象方面的资料,更全面地了解面向对象技术。

6.2 面向方面的软件测试(Aspect Oriented Software Testing)

面向方面编程是 21 世纪初期人们提出的一个相对较新的编程范型,该编程范型主要强调将软件系统特别是面向对象系统中的分布在各种类和对象中的关注点(即所谓横切关注点)分离出来,实现比原来系统更好的模块化。然而,面向方面编程给人们带来新的特性和便利的同时,也给人们在测试方面带来挑战。传统的测试方法无法有效检测面向方面编程所带来的一些故障,例如,银行系统的认证功能可以作为一个关注点分离出来,成为一个横切关注点,AOP 在源程序代码中插入切点,识别和捕获系统对相应关注点的各种方法的调用消息,如果在切点丢失了某些调用消息,认证功能就无法正确运行,从而导致严重错误,这种错误称为面向方面的切点错误。因此,尽管面向方面编程加强了软件的模块性,但与面向对象编程等其他编程方式一样,方法本身不能提供正确性保证,也可能带来

新的错误。

面向方面编程的核心就是将分散在多个模块中的一些核心关注点、中心功能，如登录功能和认证功能等，从这些模块中分离出来，横切出来的关注点独立地实现为系统的一个方面，这个方面的具体实现可以利用传统方法，例如面向对象的方法。面向方面程序运行的方式是当需要运行一个横切关注点的时候，通过切点和联合点的对接方式，相应的方面就会被调用，执行相应的方面代码，执行结束后返回。这种面向方面的编程的特点是容易维护、高度模块化和可重用、容易理解和演化，因为各项功能相对独立，不会分散(Code Scattering)、交织(Code Tangling)在程序中(如图 6-2 所示)。

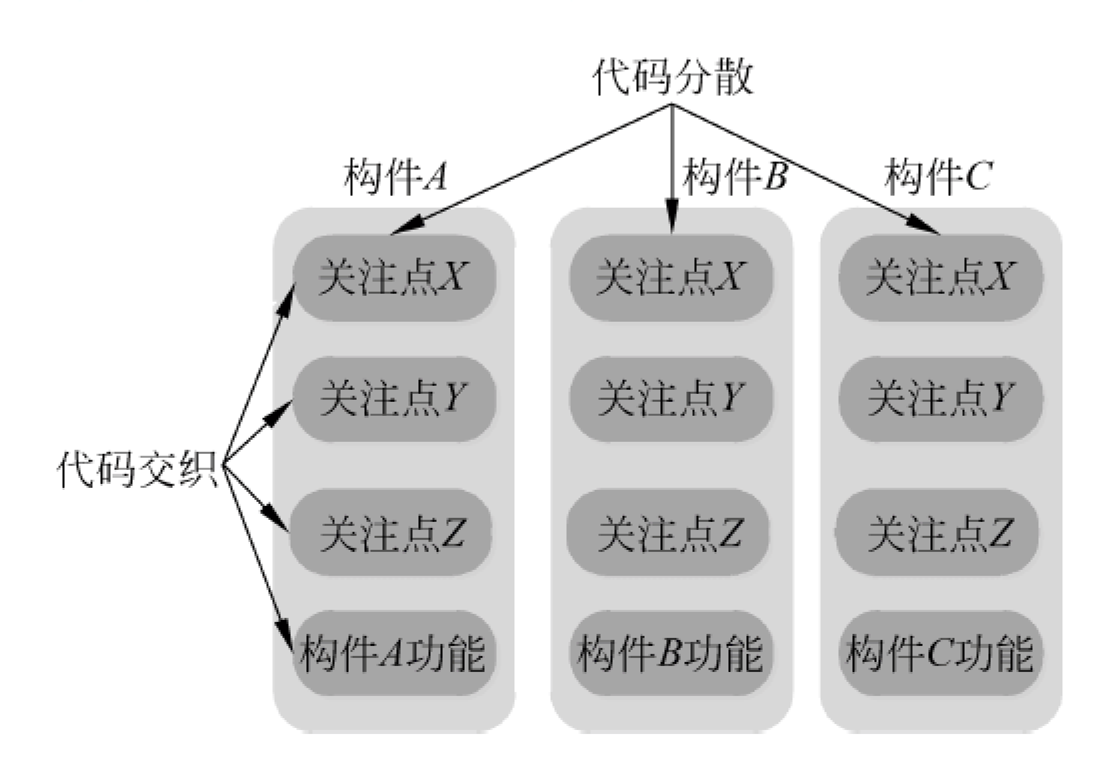


图 6-2 面向方面编程的核心示意图

Aspect J 是 Java 语言的扩展，引入了支持面向方面编程的一些概念。例如，联合点(Joint Point)表示程序中关注点执行或注入的地方；切点(PointCuts)指的是需要调用关注点的地方，包含了与关注点执行相匹配的上下文；建议(Advice)与方法一样，包含了关注点的具体代码，是关注点的具体实现。介绍(Introduction)提供系统源代码的修改；方面(Aspects)包含了实现横切关注点的一个封装，是切点和建议的组合(如图 6-3 所示)。

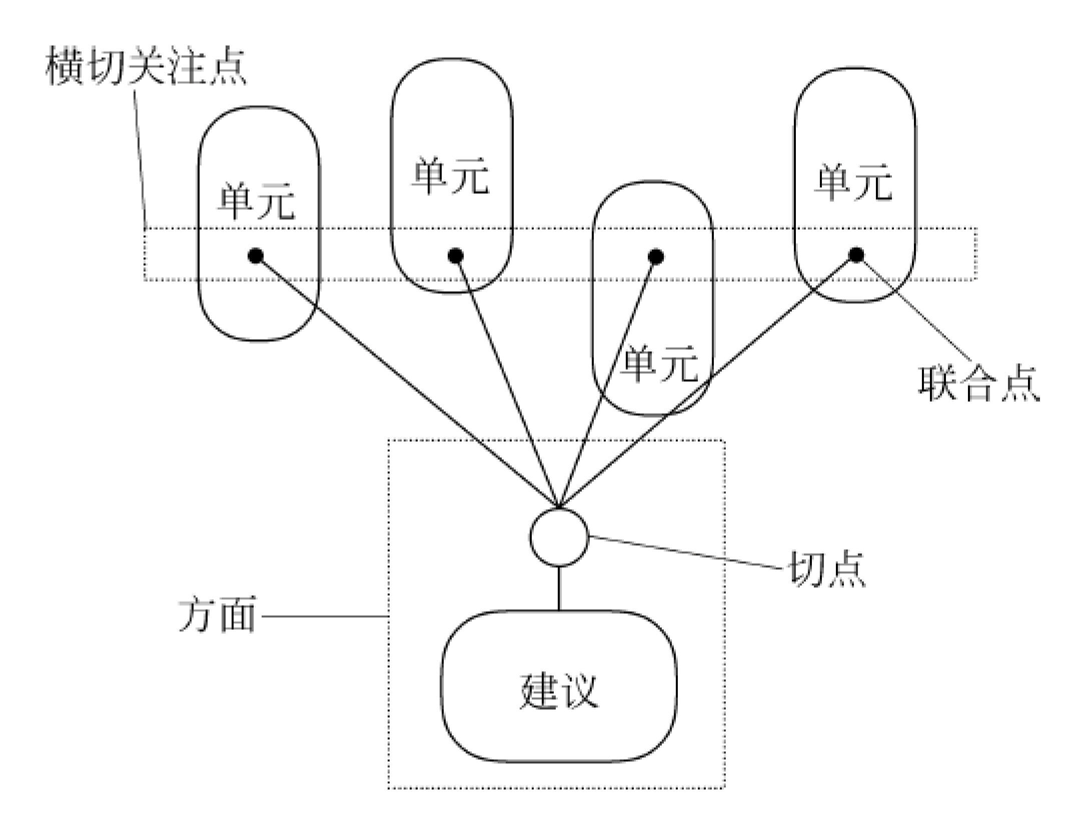


图 6-3 面向方面编程的相关概念

关于面向方面软件的测试，人们已经做了相当系统的研究，包括基于代码的测试即结构化测试或白盒测试、基于规格说明的测试即黑盒测试、基于模型的测试、基于故障和风险的测试、回归测试、变异测试以及测试用例自动生成和执行的自动化测试等。

习题 6.2

面向方面软件具有哪些特点?

6.3 面向服务的软件测试(Service Oriented Software Testing)

1. 面向服务软件的概念

面向服务的体系结构正在给当前和未来的软件工程带来巨大变化,它是一个有弹性的、高度动态的系统,该系统通过服务的动态发现和服务组合、超晚绑定、服务层协议的管理和自动协商、自治的系统重配置等机制实现。面向服务的体系结构从根本上改变了软件开发的观念,实现了软件的所有权(软件是一种产品)和它的使用(软件作为一种服务)的分离。面向服务软件的以下特点增加了测试的难度和负担:

- 面向服务的系统本质上是分布的,需要对不同的部署配置,保证其服务质量。
- 面向服务的系统中各个服务的改变是独立的,这对于该系统的回归测试会产生影响。
- 系统实现的自适应行为,如替换或增加一个新的服务,相应的集成测试必须处理这些改变的配置。
- 服务集成商和用户对于服务提供商用于广告和描述服务的信息的有限信任使得测试用例的设计很复杂。
- 服务系统通常具有复杂的多个利益方的拥有权关系,系统测试需要这些利益各方的相互协调。

因此,面向服务的系统架构不仅改变了系统的构建方法和使用方法,同时也改变了系统的测试方法,例如,服务系统使用的各种服务不是真实地集成到系统中的,而是运行在服务供应商那里,服务可以使用而不能拥有,这对服务测试意味着:系统集成商无法获得代码,服务系统中各个服务的演化是不可控的,服务系统管理员无法利用容量规划的方法阻止服务层协议的失败。

2. 目标

面向服务的软件测试对于保证服务软件的成功应用起到关键作用,它包括单个服务的测试到企业间系统联合测试等多个层面,必须同时覆盖服务功能测试和非功能性测试。

3. 原理

为了保证面向服务架构的任务关键系统的可靠性,人们采用了一些有效的方法来增强对服务系统的信心:一种方法是通过服务冗余来加强系统的容错能力,例如,可以将每个服务设计成一个可以容纳多个等价服务的容器,当系统调用时,通过投票机制选择可用服务;另一种方法是建立对服务的连续监控,任何时候,当某个服务出现问题时,监控程序启动相应的恢复程序,例如,在天气预报系统中一个温度服务不可用时,监控程序发现后,立即发现并启动一个新的温度服务,监控机制使得服务系统具有自疗自修的能力,该机制需要约简意

外事件的类型，使得监控机制可以在发现这些意外事件时，采取适当的修改动作。

传统的软件系统测试方法也可以应用到服务软件的测试中，例如，单元测试、集成测试、系统测试和回归测试等，但服务系统的动态和自适应特征使得大多数传统方法无法直接使用。例如，传统测试方法中，调用方可以精确知道被调用代码，就像在面向对象编程中，多态组件的各种动态绑定都是可知的。但在服务系统中，由于系统是在运行是在开放的服务市场中动态发现，并且是超滞后绑定，因此这一点是做不到的。

4. 方法

每当新技术出现的时候，人们就会开发一些新的软件工程方法和过程以适应新技术的发展，最典型的问题就是是否可以重用已有的方法和技术。对于面向服务架构的软件是否可以利用已有的测试方法进行测试，例如，以前针对单机系统的软件测试方法、分布式系统测试方法、构件软件测试方法和网络软件测试方法等。要回答这个问题，就必须关注服务软件系统的特殊性。具体来讲，服务系统以下特殊性限制了它的可测试性：

- 服务代码与结构的不可见性　对于用户、服务集成商来讲，服务只是一个接口，因此对服务不可能进行以代码和结构为基础的白盒测试。如果在测试中需要一些如服务操作之间的依赖关系、完整的服务行为模型等进一步信息时，除非供应商可以提供，否则测试人员必须从服务外部进行推理获得。
- 服务的动态和自适应性　在传统的系统中，调用的组件是确定的，即使在拥有多态机制的面向对象的系统中，所有可能的构件都是确定的。在服务系统中，调用的服务对象是在所有注册的服务中进行检索和选择，无法确定。
- 缺乏控制　服务组件不是物理地集成到软件系统中的，各种服务运行在各自供应商的系统中，在供应商的控制之下。这个特征使得集成商无法决定启用新服务的策略和回归测试等问题。这种情况下，可能当服务更新成新的版本时，用户和集成商未能得到评估通知。结果，在使用时，人们会遇到出人意料的系统行为，即使基本功能没有变化，服务层协议规定的性能指标可能不再吻合。
- 信任缺乏　一般情况下，服务应该提供它的行为模型等特征描述、服务性能等服务质量信息。从理论上，正是利用这些信息，集成商发现服务，理解服务的特征，最后决定采用哪个服务。然而，这些信息可能不真实，因为，供应商可能会撒谎，提供一些关于服务功能和非功能的错误信息，影响集成商的正确决策。
- 测试成本　服务系统测试过程中要大量调用运行在供应商机器中的各种服务，这需要支付相应的费用，而且调用频率过高，会引起服务拒绝。

面向服务的软件测试，一方面要开发新方法，另一方面也要使用已有方法来处理以上问题。服务系统与基于商业构件的系统非常相似，如供应商都是独立开发构件、集成商无法获得代码进行重新测试等。但商业构件是物理上集成到软件中的，而服务只是运行外部的供应商的机器中，而且服务质量是随时间发生着无法预测的变化，因此，面向服务软件需要一些独特的软件测试技术。

面向服务的软件给开发方、供应方、集成商、认证方和终端用户在软件测试方面提出了不同的需求和挑战（如表 6-1 所示）。以下分析以上利益各方在软件测试的各个层面的利弊。

表 6-1　几种典型的服务测试

	开发方	供应商	集成商	第三方	用户
功能测试	＋可以做白盒测试 ＋成本有限 ＋可以用服务的规格说明产生测试用例 －无法产生实际场景下的输入	＋有限成本 －需要服务规格产生测试用例 －只能做黑盒测试 －非代表性输入	＋在使用环境下测试服务 ＋对服务合成进行白盒测试 －对服务进行黑盒测试 －高成本	＋供应商只需少量资源(用一个第三方认证代替多个供应商) ＋评估上不会出现偏见 －只评估其他人选择的服务和功能 －高成本	服务软件自测试检查运行时是否功能符合要求 －服务没有接口允许用户测试
集成测试	可能是集成商在自己的环境下进行的测试	不做	系统在重新配置和重新绑定后需要回归测试 －由于动态绑定，要测试服务耦合调用问题	不做	不做
回归测试	服务演化或维护之后都要进行回归测试 ＋有限成本(服务可以离线测试) －不关注谁使用该项服务	＋成本有限(对服务做离线测试) －只关注服务变化，不用关注如何变化	－可能没有意识到使用的服务已发生改变 －高成本	＋与多个集成商各自测试相比，可以使用较少的宽带 －代表集成商对服务进行重新测试 －回归测试用例集合不是实际应用场景	服务软件自测试检查服务演化后是否正常工作
非功能测试	需要对供应商和消费者提供准确的非功能规格 ＋有限成本 －非现实测试环境	检查服务层协议是否符合消费者要求 －测试环境不是实际使用环境	－服务层协议测试必须考虑所有可能的绑定 －高成本 －测试结果依赖于网络配置	代表其他人评估性能 ＋减少资源使用 －非实际使用场景	服务软件自测试检查运行时性能是否满足要求

注：“＋”表示正面，“－”表示负面。

(1) 服务开发方。一方面，目标是发布高可靠的服务，服务开发方测试服务软件以期尽可能多地发现错误。开发方拥有服务的实现源代码，可以进行白盒测试。另一方面，开发方也希望评估服务的非功能属性和异常处理能力，尽管测试成本有限(开发方在测试自己开发的服务时不必支付相关费用)，非功能性测试并不是很真实，因为测试环境无法包括供应商和终端用户的真实运行环境，即开发方设计的测试用例无法反映真实的用户场景。

(2) 服务提供商。服务供应方测试服务软件的目的是确认与用户服务协议中规定的各个需求得到保证。测试成本有限，然而，服务供应方可能不能使用白盒测试技术，因为服务供应方与开发方可能不是一个组织或个人。同样，非功能测试无法反映消费者运行环境、网络配置和负载情况等，服务提供商的测试也无法反映真实的用户场景。

(3) 服务集成商。服务集成商为了确信供应商提供的服务软件满足服务集成的功能和非功能性设计要求，需要进行测试。但运行时发现和超滞后绑定使得这种测试非常困难，因为动态绑定的服务不可确定，而且，集成方无法控制不断变化的服务。所以集成商的测试需

要不断地调用供应商的服务，因此有可能浪费供应商的资源。

（4）第三方认证。服务集成商可以使用第三方认证对服务进行评估。从供应商角度，这样可以节约测试资源。为了保证容错，第三方可以代表其他人，同样从集成商的视角对服务进行测试。然而，和集成商一样，第三方认证无法在特定的服务组合中测试一个服务，也无法在同样的网络配置和环境下进行测试。

（5）终端用户。终端用户对于服务测试并无充分考虑，他们只关心这些服务在他们使用的时候能正常工作就可以了。对用户来讲，服务的动态性意味着潜在的优点，如更好的性能、更好的功能和更少的成本，但也有潜在的威胁。让服务系统具有自测试能力可以减少这种威胁，但这又意味着更高的成本和资源的浪费。例如，如果一个可以通过无线网连接的装有微型电话的服务突然通过调用几个服务启动自动测试，需要支付很多宽带使用费。

5. 实例

（1）原子服务和服务合成的单元测试

单个原子服务的测试在原理上可以认为等价于组件测试，事实上，尽管有很多相似性，但也有很多不同之处。

① 可观测性。除了开发方可以利用代码进行白盒测试，对于原子服务的单元测试只能进行黑盒测试。服务的各种模型，例如，描述状态演化的状态机模型，不仅对于测试是有益的，而且还可以进行服务发现和合成。但由于产生这些模型需要一定的技能、时间和资源等要求，开发方和供应商往往无法提供这些模型。尽管如此，人们研究一些服务规格说明的黑盒逆向工程方法，如不变式检测方法等。

② 测试数据生成。人们关于测试数据生成的研究已经非常多，例如基于搜索的测试数据生成。通常在可以利用代码的情况下，人们可以利用代码覆盖、路径覆盖等。由于无法拿到源代码，这使得测试数据的生成变得困难，对于黑盒测试生成只能根据输入输出域或一些模型进行测试数据生成。

③ 复杂的输入输出类型。已有的测试数据生成技术，绝大多数只能处理单个输入的情况。但现实世界的服务通常可以通过 XML 模式定义很复杂的输入，测试数据生成应该可以根据 XML 模式产生相应的测试数据。为服务操作产生测试数据的生成算法应该可以生成对应 XML 表示或参数操作的树或森林，为此，可以使用泛化编程。

④ 输入输出没有充分描述。要使用如等价类划分等黑盒测试技术，必须要知道每个输入的允许值范围或边界。原则上，XML 模式中定义服务运算的输入参数应该提供这些信息，但实际上，几乎很少这么做，因此，测试人员必须手工指定值的范围或可用值。

⑤ 测试成本和副作用。这是一个涉及服务所有测试活动的共同问题。单元测试应该尽可能少地使用服务调用，即测试用例集最小化问题是一个基本问题，或者服务应该提供一个测试模式，允许测试人员免费调用，而不占用相关资源，不产生副作用。

例 6-1 Bertolino 等给出一种 TAXI（Testing by Automatically generated XML Instance，自动生成 XML 测试用例的测试方法）方法和相应工具，该工具通过 XML 模式产生测试数据。虽然不是特意为 Web 服务设计的，但特别适合于服务的黑盒测试。该工具将输入域划分为各个子域，使用等价类划分方法生成测试用例。

例 6-2 白晓颖等给出一个针对服务或服务组合操作的测试用例生成和操作序列测试

框架。这个框架通过分析服务描述语言(WSDL)界面产生测试数据。我们知道，服务运算的参数类型可以是简单的 XSD 类型、集合类型(如数组)和用户自定义类型。对于简单类型，该框架通过选择一些特殊值和代表值产生测试数据；对于复杂类型，利用递归方法，直到处理简单类型。

该框架利用操作之间的依赖关系生成操作序列进行测试，一般通过 WSDL 推理得出操作间具有以下关系：

① 输入依赖关系，即两个操作有相同的输入参数。

② 输出依赖关系，即两个操作具有相同的输出参数。

③ 输入或输出依赖关系，一个操作的输出是另一个操作的输入。

例 6-3　Conroy 等开发了一个遗产软件的用户界面用来生成 Web 服务的测试数据，这种方法基于以下想法：

① 图形界面元素无非包括输入接收器、动作按钮、输出显示单元、状态检查单元。

② 从图形用户界面获得数据。

③ 将获得的数据映射为服务输入，驱动服务重新运行。

服务合成的单元测试。除了每个原子服务，服务合成时的业务流程执行语言的过程(WS-BPEL)也需要进行单元测试。开发方可以检查 WS-BPEL，所以可以对 WS-BPEL 进行白盒测试。由于 WS-BPEL 从外部看，就是一个 WSDL 描述的服务，因此，WS-BPEL 的黑盒测试与服务测试是等价的。一方面，WS-BPEL 的测试不是孤立的，需要一些桩作为搭档过程；另一方面，测试 WS-BPEL 不要访问搭档过程的可能性，允许在搭档程序开发之前对 WS-BPEL 进行测试，或者在测试阶段根本就不涉及搭档程序。

例 6-4　Li 等提出一个 WS-BPEL 单元测试框架(如图 6-4(a)所示)，该框架某种程度上与 JUnit 等其他测试框架很相似，包括以下 4 个组件：

① WS-BPEL 过程合成模型，如图 6-4(a)所示，包括待测试 WS-BPEL 过程(Procedure Under Testing，PUT)和它搭档过程(Partner Process，PP)。

② 测试体系结构，如图 6-4(b)所示，这里搭档过程 PP 用桩，称为测试过程(Test Process，TP)，控制过程(Control Procedure，CP)协调这些桩过程。因此，测试体系中用桩 TP，并测试控制结构和行为。

③ 生命周期管理，通过用户界面启动和停止控制过程 CP 和桩过程 TP。

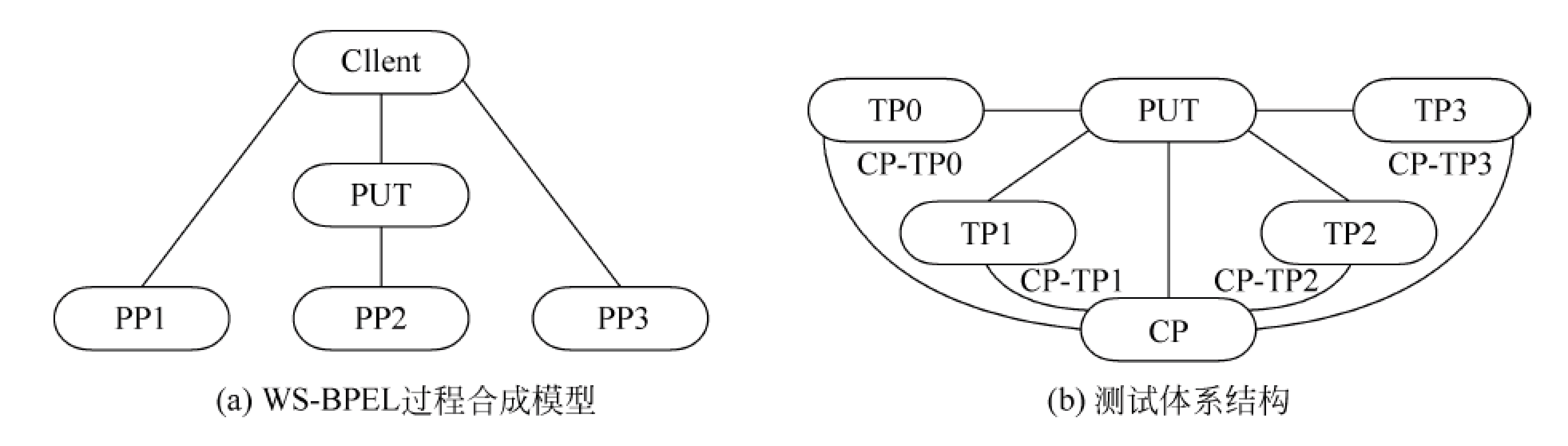


图 6-4　WS-BPEL 单元测试框架

例 6-5　Yuan 等为 WS-BPEL 测试提出了一种基于约束求解方法的测试用例生成方法，该方法主要分为 4 步：

① WS-BPEL 可以表示为 BPEL-Flow-Graph(BFG),是控制流图的一个变种。

② 遍历 WS-BPEL 的 BFG,找出测试路径,定义 WS-BPEL 活动列表。

③ 通过约束求解,过滤掉不可行路径,对于可行路径可以通过随机或手工进行测试用例生成。

④ 测试用例生成将测试数据与路径结合起来,同时通过手工给出预期输出。

例 6-6 蔡伟德等给出了一种解决测试预期输出问题的方法,他们提出同时调用多个等价的服务,通过投票机制来决定是否某个服务中含有错误,例如,某个服务的输出与其他三个不同,就认为该服务包含一个错误。

(2) 集成测试

面向服务的架构将单机应用软件的开发变成了一个应用软件由网络上分布的、多个组织开发运行的服务组成的,根据 BPCL 编写的中央控制程序进行协同工作的分布式软件。而且服务合成在一些服务不可用或一些更好服务发布时,可以动态改变以适应新的形势。因此,服务之间的交互性测试显得尤为重要,特别是集成测试。

与传统的集成测试相比,服务组合的集成测试要处理以下两个问题:①缺乏关于被集成的组件信息,这使得桩程序的开发很困难;②在测试模式中执行服务组件的不可行性,需要限制在测试过程中对服务的调用,因为这可能导致成本和资源的消耗,甚至意料之外的副作用。服务组合和搭档的动态绑定使得服务组合的集成测试更加困难,在服务组合中的调用可以动态地绑定不同地方的服务,基于不同的功能和非功能约束和目标。

在服务组合的集成测试中,一般需要测试所有可能的绑定,这就像面向对象的集成测试一样。所以,面向对象的集成测试方法可以进行借鉴和应用。在图 6-5(a)中类 A 的方法 $mc()$在调用类 B 的方法 $m()$时可以动态绑定:$B::m()$,$C::m()$或 $D::m()$。考虑到数据流,在面向对象系统中,如果方法 mA 调用方法 mB,称方法 mA 与方法 mB 之间存在一个调用连接;如果方法 mB 中使用了在方法 mA 中定义的变量,或者反之,称之为方法 mA 与方法 mB 之间存在一个参数连接。相应地,存在两种路径:

- 调用连接路径　从调用点到被调用点,然后返回(方法 mA 调用方法 mB)。
- 参数连接路径　从调用前参数最后被定义的地方到调用的过程中该参数第一次被使用的地方(定义连接),相似地,从参数在被调用方法中最后被定义的地方到返回主调方法中第一次使用该参数的地方(使用连接)。

根据以上概念,面向对象系统具有以下覆盖准则:

- 所有调用覆盖,覆盖从调用方出发,经过被调用方法,然后返回的所有路径。
- 所有定义连接覆盖,存在一条路径经过每个定义连接。
- 所有使用连接覆盖,存在一条路径经过每个使用连接。
- 所有定义使用连接覆盖,存在一条路径经过每个定义使用连接。

理想情况下,可以将面向对象软件测试的这些方法直接应用于服务合成的动态绑定测试。但存在以下一些问题:

① 在服务组合测试中,参数连接准则有所不同,因为搭档过程在被供应商服务器调用时,只能作为黑盒实体。换句话说,在调用方定义的参数,其可见的使用是对搭档过程的调用,而在被调用过程中定义的参数只能在返回点可见。

② 对应于一个抽象服务,可能的具体服务是无法预知的。一些在设计时可用的服务在

实际运行时可能无法使用，如果在开放的市场环境下利用动态发现，运行时可能调用一些新的服务。

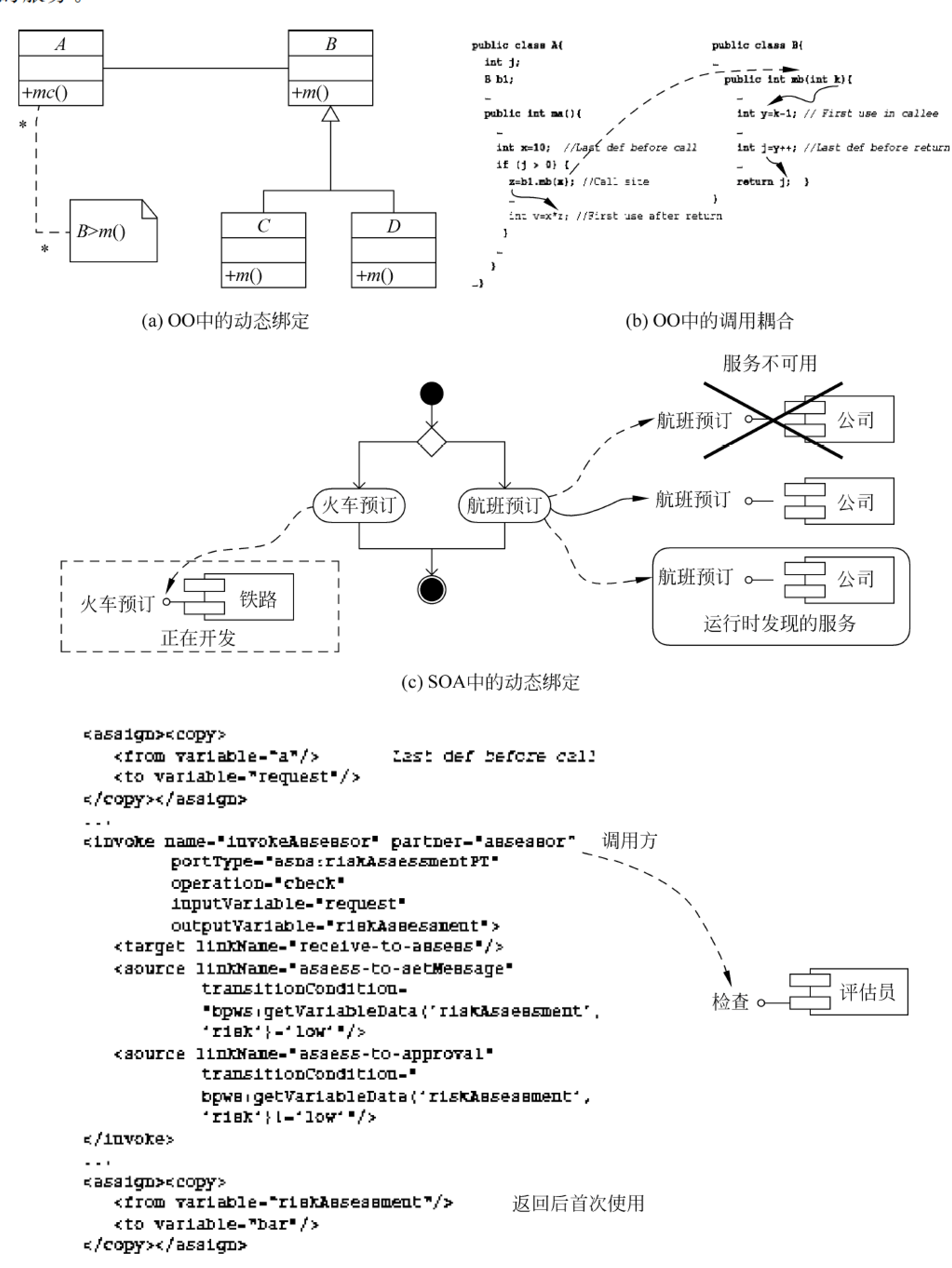


图 6-5　面向对象(OO)与面向服务(SOA)软件中的动态绑定和调用耦合

③ 对于所有可能的绑定获得以上覆盖准则是过于昂贵的，一般也不大可能。

例 6-7 蔡伟德等定义了一个服务集成测试的框架，称为 Coyote，支持测试执行和测试场景管理。Coyote 工具包含两个部分：测试主和测试引擎，测试主从 WSDL 规格说明产生测试场景，而测试引擎与被测试的服务进行交互，给测试主提供跟踪信息。

例 6-8 Bertolino 等提出一种方法确保在 UDDI 中注册的服务之间的互操作性，任何已经注册的服务都可以与正在注册的服务协同工作，他们提出 UDDI 注册要从一个被动的服务目录转变为一个负责验收审计新服务的权威。

集成测试中的服务调用也关注服务层协议，由于工作流中每个抽象服务可以和很多具体服务绑定（相当于具有不同服务质量的功能点），有可能存在某些绑定的组合不满足服务层协议的规定。

（3）回归测试

服务系统不同于传统应用软件的一个重要问题是服务集成商对服务缺乏控制。服务集成商选择服务，将它们集成到服务系统中，并假定这些服务在使用时能够实现它们的功能和非功能特性。然而，事实不是这样的，正如其他系统一样，各种被集成的服务也要不断地维护和演化，而且，维护和演化的策略不在集成商的掌控之中，服务的变化会影响正在使用它的软件。这一点，不同于组件系统，组件系统将组件拷贝集成到系统中，当原组件进行维护或者故障修复等工作时，系统仍可以使用旧的组件。服务的维护和故障修改存在以下场景：

① 一些改变不需要修改服务界面或规格说明，即供应商认为是很小的改动。结果，这些改动对使用该服务的所有人都是不可知的。

② 修改虽然不影响服务功能，但影响服务质量，服务供应商没有做相应文档，结果使该服务的系统的服务质量会受到影响。

③ 一个服务可以是由其他服务复合而成，因此，服务修改的影响可以在不同供应商之间传播，由于这些修改和受影响的行为之间存在一定距离，即使对这些修改进行了文档和告知，集成商也不大可能得到这些信息并作出相应反应。

以上场景提出了一个新的需求，即集成商要周期性地重新测试正在使用的服务，以确保这些服务仍然满足预期的功能和非功能特征。为实现对服务的周期性测试，Di Penta 等提出在服务描述中增加提供已有测试用例的方面（facet），该方面以基于 XML 的功能和非功能断言形式进行描述。一个方面是一个用来描述服务特定性质的 XML 文档，例如，服务界面（此时对应 WSDL 界面）、服务质量（QoS）、用于与服务潜在用户谈判的服务层协议（SLA）等。利用方面描述服务的方法将原先的 UDDI 注册方法扩展为基于 XML 的包含多个方面的注册方法。然而，其他方面只要连接服务的 WSDL 描述，不需要定制的专有的注册。图 6-6 给出的 SOA 概念模型片段就是利用方面来描述服务性质的，一个方面就是用一种语言描述的方面规格说明，如用 WSDL 描述的界面、服务协议描述的 SLA。在图 6-6 中可以看出，一个服务可以被描述为签名、操作语义、服务质量、测试用例等多个方面。

当一个服务集成商发现一个服务并想使用的时候，他就会下载这个服务的测试用例方面，用其中的测试用例集检查该服务是否具备预期的功能和非功能特性。服务的这个测试用例集包含了开发方对该服务软件的理解。

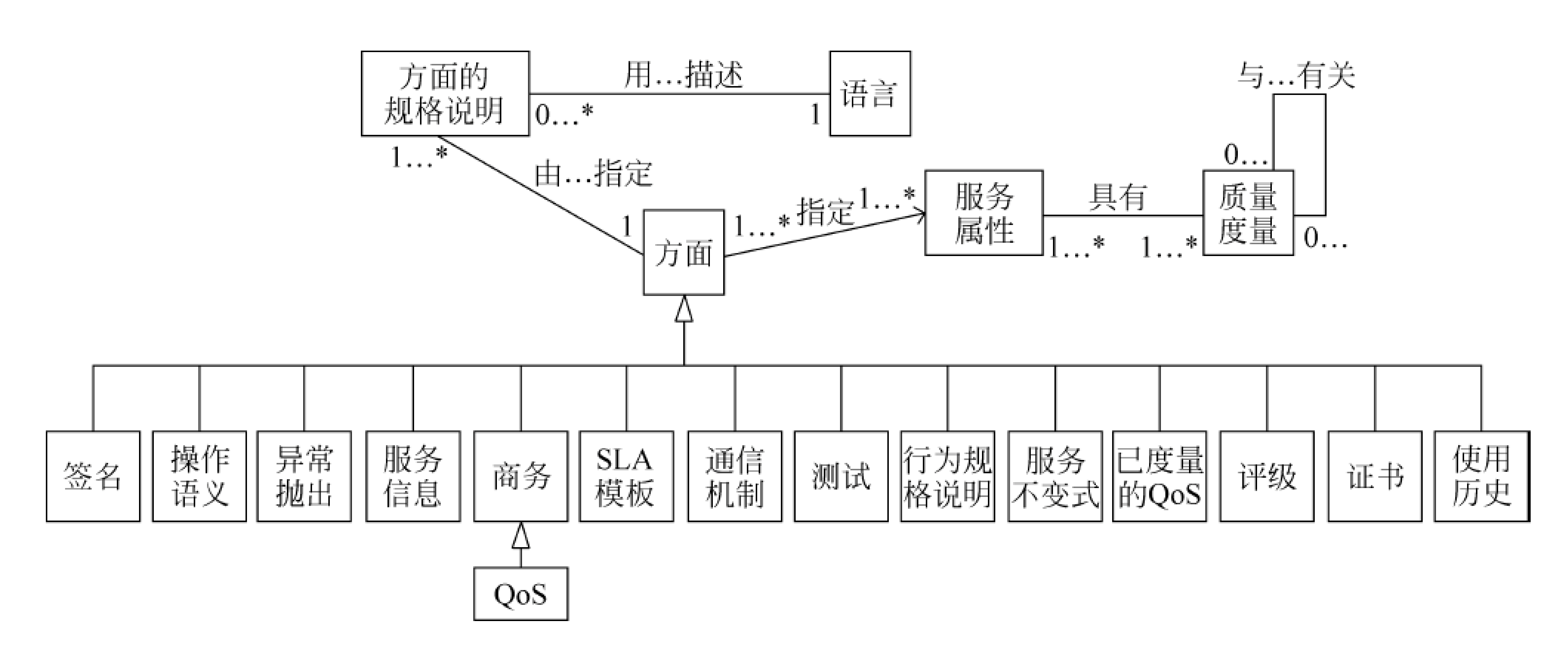


图 6-6　服务的各个方面的规格

例 6-9　图 6-7 描述了测试用例发布和回归测试过程的几个可能场景，该场景涉及服务供应方(Jim)、服务集成方(Alice 和 Jane)，用以解释提出的回归测试方法的测试能力。

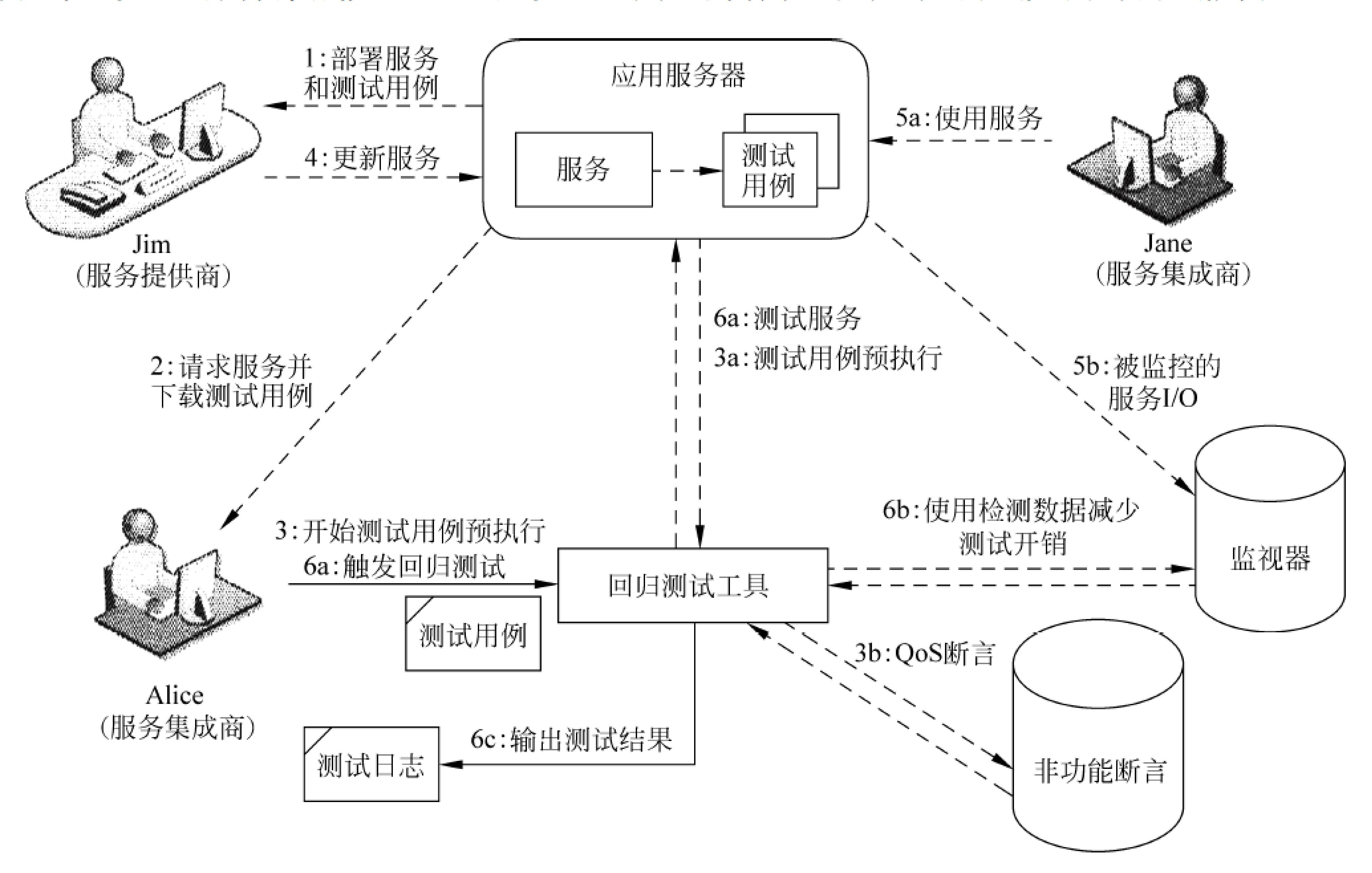


图 6-7　服务回归测试：测试用例生成与执行过程

① Jim 部署了一个餐馆预订服务，该服务允许用户查找餐馆，获得餐馆信息并检查其可用性，测试该餐馆预订服务的测试用例也一起发布。

② Alice 发现了这个服务，查看了服务层协议并下载了测试用例集；她可以增加一些测试用例，然后对这些测试用例进行执行，度量该服务的非功能特性，在与供应方服务层协议达成一致之后，她将测试用例和测试阶段获得的服务质量断言保存了起来。

③ Alice 定期使用该服务，直到：

④ 一段时间之后，Jim 更新了该项服务。在新版本服务中，返回的餐馆标识从四位变

成五位，同时由于配置变化，该服务无法在2秒之内响应。

⑤ Jane经常使用该服务，没有发现任何问题。她使用一个域可视化工具可以看到由5个字节组成的餐馆名字。同时Jane对该服务的交互操作是受到监控的。

⑥ 由于服务的变化，Alice决定对其进行测试。Jane执行该服务的监控数据可以用来减少测试用例执行次数。最后报告功能测试和非功能断言的成功和失败信息。

支持服务回归测试的方面模型可以由服务开发方或测试人员手工建立，或者在系统单元测试阶段产生。利用测试用例集建立供应方和用户之间的合同，人们可以通过运行测试用例来检查服务是否都可以达到合同中规定的功能和性能。测试用例约简是服务组合回归测试的一个重要问题，目前主要是将传统方法应用过来，例如，利用控制流图(CFG)中已改变的地方选择适合的测试用例进行回归测试。但事实上，几乎没有一个服务供应商会提供服务的CFG。

(4) 非功能测试

面向服务的软件系统中，非功能性质的测试是非常重要的，主要有以下原因：

- 服务供应商和客户之间规定了服务测试协议，保证了客户在使用某些功能时的服务质量。然而，在有些条件下，由于用户出乎意料的输入、或难以意料的服务负载，这样的服务质量可能难以实现。
- 由于缺乏健壮性和可恢复性，对于一些非正常行为，服务可能在供应方或集成方产生难以意料的副作用。
- 服务是面向网络开放的，可能受到安全攻击。

人们提出了很多不同方法，对服务软件的非功能特性的很多不同方面进行测试：

① 健壮性测试。Martin等提出一种自动的服务健壮性测试工具，该工具首先根据WSDL自动生成一个服务客户端，然后，任何面向对象程序的测试用例生成工具都可以用来执行服务健壮性测试，特别地，Martin等使用了JCrasher(产生JUnit测试用例的工具)。他们使用这个健壮性测试工具对Google搜索和Amazon等服务进行了测试，虽然没有能够发现一些重要问题，但实验显示有时服务会被挂起，这可能意味着存在某些故障。

要保证服务的健壮性，对一些异常行为不仅要能够正确反应，而且要能做出正确处理，实际上，错误恢复代码很少能得到适当的测试。人们提出了异常捕获连接的概念，该连接用来建立错误敏感操作和程序中错误捕捉块之间联系，人们定义了对这种连接进行覆盖的测试准则。

② 服务层协议(SLA)测试。这种测试的目的是找出服务无法提供服务层协议指定服务质量的条件，在给客户承诺服务质量之前，供应商应该尽量减少这种情况的发生。Di Penta等提出利用遗传算法对原子服务和服务合成进行服务层协议测试。导致服务软件无法达到承诺的服务质量的因素很多，主要包括用户输入、抽象服务最终绑定的具体服务、网络配置和服务器负载等。遗传算法产生用户输入与绑定服务的组合导致服务软件无法达到服务层协议要求。

对于原子服务，使用遗传算法产生输入，通过监视服务调用过程中的服务质量，如响应时间、吞吐量和可靠性等，当然也可以观察领域相关的服务质量。监视取得的服务质量作为生成的测试用例的适应值，来驱动通过交叉变异来生成新的测试用例；那些越接近违反服务协议的测试用例，其适应值越高。为了能够利用遗传算法生成服务输入，将服务输入根据

XML 模式表示成树或森林结构，这样就可以通过交叉和变异操作来演化服务的输入空间。

对于服务组合，特别是在动态绑定一些具体的服务时，遗传算法产生输入与绑定的组合，使其导致服务协议无法达到。实际上，同样的输入，不同的绑定会导致不同的服务质量。图 6-8 是一个图像处理工作流，我们作如下假设：

- 服务供应商保证对消费者的响应时间不超过 30ms，图像结果大于等于 300 点/英寸。
- 用户提供一个尺寸小于 20Mb 的图片作为输入(这是服务协议的前提)，posterize = true，dim1 = dim2，nsharpen = 2。
- 抽象服务分别绑定为 ScaleC、PosterizeC、SharpenB 和 GrayA。

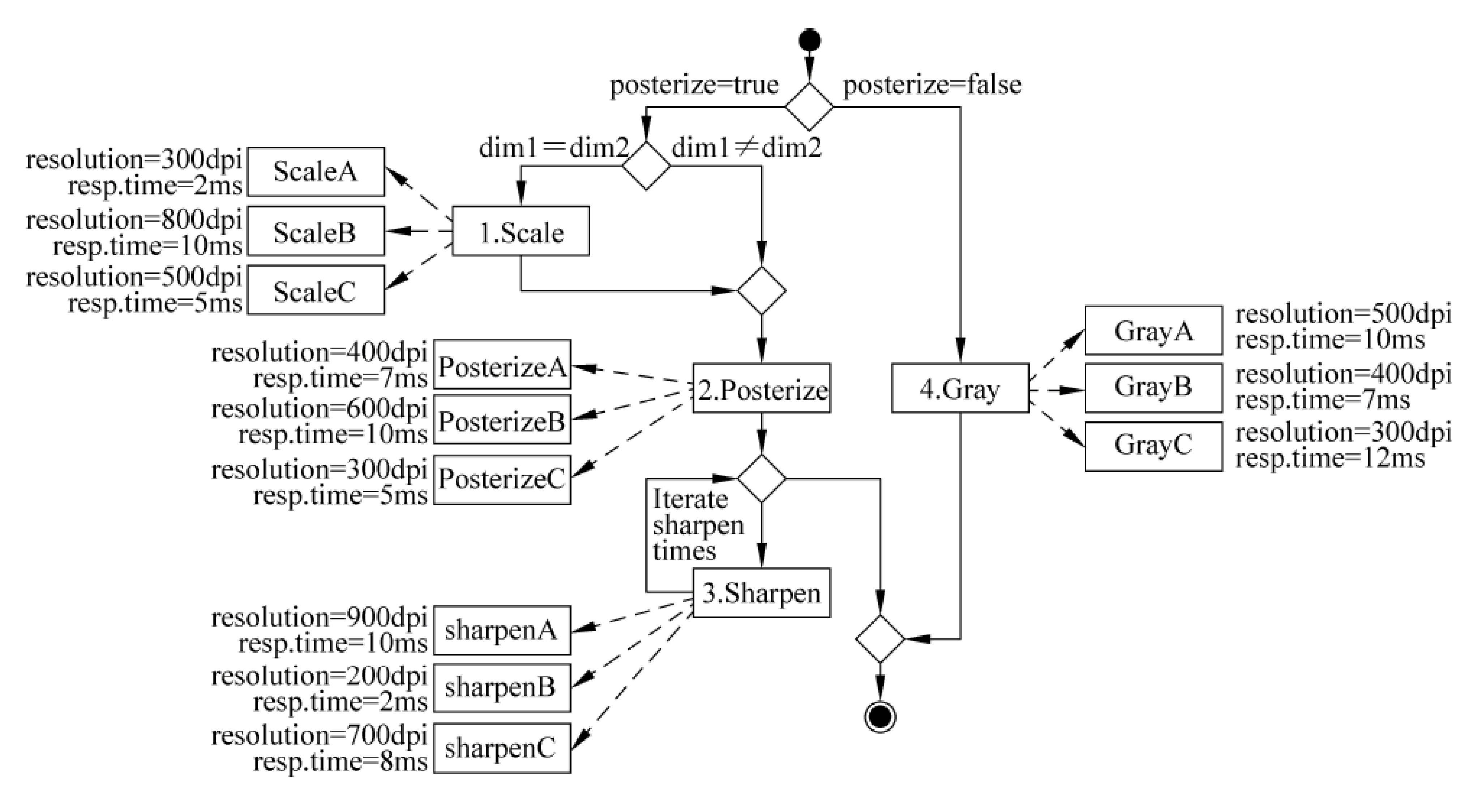


图 6-8 服务合成的服务层协议测试实例

在这种情况下，当响应时间有较低的界 24ms 时，约束可以满足，服务合成产生的图片为 200 点/英寸时，就对应被调用服务的最低质量保证。另外一种情况就是，产生输入与被调用的服务间的各种组合时，就可能出现服务协议达不到的场景。

③ 可靠性测试。上面介绍的方法有一个特殊的目的，即产生服务输入(或绑定)使之不满足服务协议。更一般来讲，对一些关键的商业服务软件，产生可以导致如下行为的输入是非常有用的：

- 安全问题，例如允许未授权用户访问敏感数据；
- 服务挂起；
- 意料之外的，无法处理的异常；
- 服务反应中无法观察的行为。

为了产生有效输入，人们提出基于 SOAP 消息的数据扰动方法，不需要修改源代码，就能改变服务软件的内部状态。该方法首先向服务器发送一个请求消息数据，观察服务器反应，然后每次修改原始请求消息数据，并重新发送该数据，观察每次服务器反应，通过比较发现问题。数据扰动方法(Data Perturbation，DP)包括 3 种形式：数据值扰动方法

(Data Value Perturbation,DVP)、远程过程调用(RPC)通信扰动(Remote Communication Perturbation,RCP)和数据通信扰动(Data Communication Perturbation,DCP)。

数据值扰动方法(DVP)根据服务输入参数的 XML 模式对输入参数进行修改,这种方法类似于边界值测试方法。例如,对于数值型变量,考虑最大值、最小值和 0;对于字符串,考虑最大长度串和最小长度串等。

远程过程调用通信扰动(RCP)修改消息和通信数据,特别是程序中使用的数据和数据库中的输入数据。例如,如程序中使用的数据进行变异,可以将一个数值 n 改变为 $1/n$, $n\times n$, $-n$ 或 $|n|$ 等,还可以改变参数顺序。

例 6-10 如果服务有 2 个输入参数用户名和密码,然后使用查询语句检查数据库中是否存在这个用户名和密码。

```
SELECT username FROM adminuser
        WHERE username = 'turing'AND password = 'enigma'
```

非授权化扰动机制将字符串'OR '1'=1'附在用户名和密码后面,结果查询语句变为:

```
SELECT username FROM adminuser
        WHERE username = 'turing' OR '1' = 1'AND password = 'enigma' OR '1' = 1'
```

这个查询总能通过。

另外一个数据扰动的例子是:给定一个 SOAP 消息中包含描述书的数据结构。

```
<book> <ISBN> 0 - 781 - 44371 - 2 </ISBN> <price> 69.99 </price> <year> 2003 </year>
</book>
```

具体的数据扰动可以是发送一个:①空的数据结构实例;②允许的实例个数;③消息中的实例的几个副本;④从消息中将实例去掉一个。以下是在消息添加了一个副本的例子:

```
<book> <ISBN> 0 - 781 - 44371 - 2 </ISBN> <price> 69.99 </price> <year> 2003 </year>
</book>
<book> <ISBN> 0 - 781 - 44371 - 2 </ISBN> <price> 69.99 </price> <year> 2003 </year>
</book>
```

以上的数据扰动是有用的,可以检查在消息中有副本时,反馈消息中是否有不同的行为。

例 6-11 Looker 等提出一种故障植入技术来评估服务可靠性。他们将基于 SOAP 的网络消息分解,然后往这些消息中注入一些错误。他们将服务的错误模型分为 4 种:物理错误、软件错误(程序或设计错误)、资源管理错误(如内存泄漏等)和通信错误。通信错误包括消息重复、消息遗漏、额外附加消息(攻击消息)、消息排序的改变和消息的滞后等。

当植入一个错误时,服务可以有如下不同反应:①服务崩溃;②网络服务器崩溃;③服务挂起;④服务产生破坏的数据;⑤服务产生的数据出现丢失、重复和延迟等现象。错误植入的目的就是要确定在什么程度下服务仍能够对植入的错误输入进行正确反应,具有正常的异常处理能力和恢复能力,而不会出现以上几种错误。

(5) 提高可测试性

面向服务软件的固有特点使得它的可测试性具有很大限制,根据蔡伟德等人的工作,服

务可测试性受限于以下因素：

① 服务的可访问性，即源代码的可访问性、二进制可访问性、模型可访问性、签名可访问性等。

② 描述服务的数据系列，这些信息是服务的一个重要方面，因为集成商可能不信任这些数据。

③ 面向服务软件的动态性。

为了提高服务软件的可测试性，人们提出了很多方法。

例 6-12　通过扩充 WSDL 来提供更多信息来加强服务的可测试性，具体来讲，增加以下信息：

① 输入输出依赖关系。利用 WSDL 模式引入一种输入输出类型，用以描述服务操作的输入输出关系。

② 调用序列。在服务测试中的一个重要问题就是一个服务可能委托另一个服务执行某个任务，测试人员可能需要知道这种关系，利用 WSDL 模式定义一种调用依赖类型来描述调用者和被调用者之间的关系。

③ 功能描述。建立功能描述的层次结构可以增强黑盒测试的效果。

④ 操作序列规格说明。合法操作的调用序列可以用范式的形式表示，例如，OpenFile·(ReadLine|WriteLine)*·Close 表示打开一个文件，对文件进行零次或多次读或写，最后关闭文件。

蔡伟德等也提出扩充 UDDI 注册机制，增加测试功能，在 UDDI 服务器中除了存储 WSDL 的规格说明之外，还要存储测试脚本。

Heckel 等提出使用图形转换系统来测试服务，他们假定服务注册中存储了这些信息，服务供应商利用 WSDL 等接口描述语言对服务进行描述，用图形转换规则来描述服务行为。他们还提出利用合同设计来加强服务测试。当服务提供商承诺为消费者提供一项所需要的服务时，供销合同上将描述这项功能。他们认为服务合同应该包含 3 个层次：

① 模型层，供人理解的。

② XML 层次，可以集成为已有的 WSDL 等服务标准。

③ 实现层，可以被一些工具如 Parasoft Jtest 等用来产生测试用例。显然，这几个层次之间需要相应的映射。

在服务中补充测试用例，作为合同的一部分来保证该服务具有相应的功能和非功能特性，这种做法是非常有用的。

服务测试是高度动态的，如运行时发现和绑定，这就需要一个测试中间代理，可以将测试实现与测试定义剥离，将待测试系统与测试环境分开，这样当待测试服务改变接口操作、改变绑定和运行步骤时，测试中间代理就可以进行运行时绑定和重新配置。

服务系统测试的一个重要问题就是需要一个用于评估待测试服务或服务组合的服务质量的测试工具，这些工具可以在不增加成本或副作用的前提下，准确度量服务质量。

习题 6.3

面向服务软件有什么特点？发展面向服务软件具有哪些重要意义？

6.4 构件软件测试(Component Based Software Testing)

在过去几十年中,软件开发方式适应时代发展要求已经发生了很多变化,特别是随着时间和资金压力的增加,出现了基于组件开发的概念。基于组件的开发将软件项目分解后外包给其他开发组织,最后将这些第三方组织开发出来的组件组装成完整的软件系统,以达到保证软件质量、提高效率、降低成本的作用,是人们用来克服软件危机的一种重要途径。软件组件是一个可以重复使用的程序单元,它具有详细的接口描述、明确的上下文依赖和确定的质量要求,可以独立部署,可由第三方开发。

软件组件式开发方法来源于传统工程领域,在传统的工程领域中,产品的生产先从零部件开始,然后逐步组装成正式产品。在软件开发中这样做的好处是软件组件可以多次重用,组件开发的投资可以摊销到多个应用软件当中,既提高了资源利用率,又可降低成本。而且,人们一般更倾向于解决好规模适当的小问题,当软件被划分为组件时,每个组件可经过精心设计和定义,从而容易得到很好的质量。在构建组件系统过程中,合成是一个关键技术,人们可以在不关心组件本身细节的基础上,集中精力合成新的软件系统。

高效的组件开发的一个挑战是组件的粒度和相互依赖性必须在开发生命周期的早期就要进行适当控制。组件一般在供应方进行开发测试,组件的开发方很难充分考虑组件所有可能的应用场景,这有可能导致组件在新的应用中出现问题,组件的开发方和使用方会从不同角度,利用不同标准对组件进行测试,这是组件测试技术的最大问题之一。人们分析了跟测试相关的组件特征,主要包含以下 4 个方面:

(1) 组件可观察性。组件的操作行为,输入参数和输出可以被观察的容易程度,组件接口的设计和定义对于决定组件可观察性起着主要作用。

(2) 组件可追踪性。组件跟踪其属性状态和行为状态的能力,前者称为行为可追踪性,组件具有跟踪其内部和外部行为的能力;后者称为跟踪可控制性,组件具有调整跟踪功能的能力。

(3) 组件可控制性。组件的输入输出、操作和行为的可控制难易程度。

(4) 组件可理解性。组件信息可以提供多少,并且这些提供的信息质量如何。

对于那些非常重要的组件,必须要认真设计测试计划,进行科学系统的测试。例如,对于那些经常重用的组件需要进行广泛的测试;对于那些领域组件不仅要对其正确性进行测试,还要测试其文档描述的准确性;商业组件不仅要作为可重用组件进行测试,还要测试其作为潜力资源的可靠性。组件测试的目标是检查组件的实现是否与规格说明一致,是否充分实现了所有需求。组件测试有很多策略,主要包括以下几个方面:

(1) 组件认证。通过一套组件认证方法,确认组件的高可靠性和对相应系统的高质量性。通过黑盒测试、故障植入测试和系统容错测试等对组件的质量进行评估。

(2) 组件综合信息方法。所谓组件综合信息是指组件供应商提供的关于组件的各种信息,每种信息都有特别的格式和标记。组件综合信息方法利用这些信息对组件进行分析和测试。组件综合信息是由组件供应商利用分析技术收集的信息,这些信息提供给用户,用户就可以不再需要源代码,组件用户应该可以在组件中查询这些信息。组件中每个操作的输入空间被分成若干子域,每个子域都附有说明,这些说明可以作为测试组件时的查询信息。

(3) 利用组件的 UML 模型进行测试。利用 UML 序列图和协助图模型设计测试用例检测组件间交互可能存在的问题,UML 测试模型包含表示集成组件的结点和表示组件间交互的消息流。测试用例的选择根据图模型测试标准来进行。

习题 6.4

基于组件的软件系统与传统软件系统有哪些不同之处?

6.5　Web 应用软件测试(Web Testing)

1. 概念

对于 Web 应用软件来说,测试也是为了发现软件的错误并最终修正错误而运行软件的过程,但由于基于 Web 的系统和应用存在于网络上,并且和很多不同的操作系统、浏览器(或其他界面设备如 PDA、手机等)、硬件平台、通信协议、后台应用进行交互,因此对于 Web 应用软件的测试具有更大的挑战。

2. 目标

Web 应用软件如果出现质量问题,将严重影响公司形象,动摇终端客户或用户的信心,Web 应用软件测试就是为了发现 Web 应用中的内容、功能、易用性、导航、性能、容量、安全性等方面的错误,为了系统有效地检测这些可能的错误,需要使用包括静态评审和动态执行测试在内的各种测试方法。

3. 原理

为了理解 Web 测试的目标,必须考虑影响 Web 应用质量的 10 个方面:

(1) 内容(Content)是在语法和语义级别的评估。在语法级,要评估基于文本的文档中的拼写、标点、语法等;在语义级,要评估信息表现的正确性、整个内容对象和相关对象的一致性、无二义性。

(2) 功能(Function)测试是要发现与客户需求不符的错误。每个 Web 应用功能要评测其正确性、不稳定性(Instability)、与实现标准的符合性(如 Java 或者 XML 语言标准)。

(3) 结构(Structure)评估要确保正确发布了 Web 应用的内容和功能,并且是可扩展的,支持新内容和新功能的增加。

(4) 易用性(Usability)测试是要确保对每一类用户都要有相应的界面支持,用户要能学习和应用所有的功能、所有需要的导航句法和语义。

(5) 导航(Navigability)测试要确保所有的导航句法和语义都被测试,从而发现有关导航的任何错误,例如死链接、不合适的连接、错误的链接等。

(6) 性能(Performance)测试检测各种操作条件、配置、负载变化的情况下,系统在响应用户交互和处理极限负载时,性能没有出现不可接受的退化。

(7) 兼容性(Compatibility)测试通过在服务器和客户端不同的配置情况下,执行 Web 应用,目的是发现跟某种配置相关的错误。

(8) 互操作性(Interoperability)测试是检查 Web 应用与其他应用或数据库进行交互。

(9) 安全性(Security)测试是评估潜在的易受攻击的弱点并尽量发现这些弱点,任何成功的渗透都是安全上的漏洞。

(10) 运行环境(Configuration)测试主要检测 Web 应用对于各种复杂的软硬件环境能否正常工作,很多错误可能只在某些特定的环境中出现。有些错误出现在运行过程中的动态环境中,如实时资源负载、网络拥挤等。

4. 方法

Web 应用是一种特殊的软件系统,因此,Web 应用测试方法与一般的软件测试方法类似,主要包含以下步骤:

(1) 评审 Web 应用的内容模型以发现内容错误,评审 Web 应用的接口模型及设计模型以发现导航错误。

(2) 测试用户界面并选择每个功能模块进行单元测试。

(3) 在各种不同配置环境下实现 Web 应用,并测试每种配置的兼容性。

(4) 由一定数量的最终用户测试 Web 应用,他们和系统的交互结果用来评估 Web 应用的内容、导航有效性、易用性、可靠性和安全性等各种性能。

5. 实例

我们每个人应该都很熟悉网站,因为网络已经渗透进社会生活,网站是网络世界的窗口。人们对于网站质量的期望已经超过了对一般商品质量的要求。对于普通商品,人们一旦购买了它,只要质量说得过去,人们就会使用它。然而对于网站,如果质量有问题,例如,在竞争者网站只需要几秒钟就能做的事情,而自己的网站却要几分钟,甚至更长时间,客户就会流失到竞争者那里。因此,网站质量不仅会给公司造成客户的流失,更重要的会严重地损害公司形象。毕竟,一个公司如果网站都做不好,怎么可能可以赢得消费者的信任呢?因而网站测试具有非常重要的意义和价值。

本书已经介绍过很多通用的软件测试技术,可以利用 Windows 的写字板、计算器和画图软件试验使用这些方法。一个非常时新的、非常现实的可以综合使用各种测试技术的测试对象就是网站测试。网站测试涉及软件测试的很多方法和技术,如配置测试、兼容性测试、可用性测试、文档测试、安全性测试,如果是世界范围内可以访问的网站,还包括国际化和本地化测试。或者说,网站测试可以使用所有的黑盒测试、白盒测试、静态测试和动态测试技术。一个网站的页面通常都是一些文字、图片、声音、视频和超链接组成,如图 6-9 所示的扬子晚报的新闻网站的主页,从这个主页上可以看到,不同颜色、格式和大小的文字;图片和照片;带有超链接的文本和图片;滚动的广告;具有下拉选择框的文本;用户可以输入数据的输入域等。

网站具有很多不是非常明显的功能,这些功能使得它更加复杂,例如,网页具有可以改变的信息在屏幕上显示的布局;允许用户定制选择他们想看的信息;动态的下拉选择框;可以动态改变的文本;关于网页显示的选项信息;对各种浏览器和硬件配置的兼容性;各种隐藏的格式、标签以及未来增强网页可用性的嵌入信息等。特别是对于一些电子商务网



图 6-9　扬子晚报网主页

站，具有很多安全防范措施。如果读者是一个职业的软件测试人员，可能会跃跃欲试，对这个网站进行测试。

下面将首先考虑对这个网站进行黑盒测试。网站给人们提供了一个非常好的实践场所，可以不去拿别人的程序来测试，只需要在网站的各个页面上跳来跳去，开始人们的测试。整个网站就是一个黑盒子，人们不知道它如何工作，甚至不知道规格说明，只有面前的这个网站。

如果把网站的每一页处理成一个状态，把网页之间的链接处理成一条有向线，由此可以建立这个网站的结构图。对于文本的测试，可以利用“文档测试”一节学习到的方法进行。对于文本中出现的通信方式进行测试，看看是否可以联系。检查版权申明的内容和日期；检查有些动态说明信息是否可以正常显示，例如，当用户鼠标移到一个图片或文本，就会弹出一个文本框，检查这种文本框是否在各种配置下都工作；检查当浏览器窗口的高度和宽度变化时，是否可以正常显示。

对于网页中各种超链接，检查是否可以链接到相应的网页，检查链接标记是否明显（一般是加下划线，或当鼠标移到时变成其他形状），检查是否有无法访问的页面。对于页面中的各种图片，要检查是否可以正常显示，查看页面加载时的速度，比较测试局域网和远程网网页的图片显示效果和速度。检查页面中表格，对每个输入域进行输入测试，检查它们是否能接收正确数据而拒绝不正确数据。

一个网站就是一个软件，它可以包含一个点击计数器、滚动的文本、不断变化的广告、内置的搜索引擎。设计测试时，要首先找出每个网页具有的功能，把每个功能当成一个独立的软件，并利用已有的软件测试方法和标准来单独测试。如果一个网页处理数据，就要对处理的数据进行边界值分析、等价类划分等处理。如果测试人员懂得使用 HTML，了解网页工作原理，那么就能做一些灰盒测试，即介于黑盒和白盒之间的测试，根据自己的兴趣，做一些

更有效的测试。

有很多网络编程语言和技术，例如，DHTML、Java、ASP、ActiveX 等，作为测试人员无须成为这些语言和工具的专家，只要能够熟悉和阅读，并能设计相应的测试用例就可以了。对于利用这些语言和工具产生的内容，如动态内容、数据库驱动的网页、编程产生的网页，以及服务器性能和下载等方面的测试都需要做一些白盒测试。特别是一些金融、医疗和电子商务网站，这些网站包含一些极为保密的用户资料，网站安全性极为重要，为了进行充分的安全性测试，也需要从内部工作原理出发做白盒测试。配置兼容性测试和可用性测试是网站测试的一个重点和难点，可以使用关于软件配置测试、兼容性测试和可用性测试的一般方法和技术。网站测试存在着巨大的工作量，纯粹手工完成，几乎是不可能的，因此要充分使用各种自动化测试工具。网站系统只是众多的软件系统中的一种，正是由于多种多样的软件系统，为软件测试提出了没有止境的问题，伴随着各种新技术的出现，新软件的发布，越来越多的软件测试问题会不断地被解决。

习题 6.5

1. 网站的页面有哪些基本元素？
2. 对一个网站一般要从哪些方面进行测试？

6.6 普适计算环境下的软件测试（Pervasive Computing Software Testing）

普适计算使得计算无时不在、无处不有，并使得计算实体隐藏在场景之后，人们只需要集中精力做好日常工作，无须分散精力于操作各种技术和设备。普适计算环境通常将物理环境和逻辑部件整合在一起形成一个聪明的或者主动的空间，在这个环境下计算实体应该能够感知被称之为背景或上下文的环境属性。

普适计算软件是实现普适计算的关键技术，这种软件是上下文敏感的，可以根据环境因素的变化主动地进行适应性调整。当环境发生变化时，普适计算软件可以自动地选择资源和通信实体，选择相应的执行算法。人们已经提出并开发了很多不同的普适计算软件，一般地，这类软件系统集成了环境相关的数据公式、事件、编程框架，以及上下文信息的获取和通信机制。不同的普适计算软件因不同的研究目的和观点，在模型和规格上有所不同，但绝大多数系统采用了基于中间件的系统结构。中间件技术可以将不同设备、人机之间以及外部环境的通信和数据处理封装起来，形成统一接口，这样使用起来非常简单灵活。基于中间件的结构，普适计算软件其实是一组软件的集合，这组软件就是中间件，通过这些中间件与外部环境的交互通信以及提供的服务，普适计算软件可以及时广泛地收集外部信息，很好地调整自身行为，为用户提供准确的服务。

普适计算软件历史较短，但应用广泛，没有统一的模型，传统的测试技术难以充分发挥作用，普适计算软件测试面临的挑战主要体现在 3 个方面：

首先是上下文背景是挥发性的和瞬态的，以流的形式从普适环境中传播，程序执行过程中，由于环境的自发的互操作，背景值可能不断变化。如果程序是利用中断驱动的语言实现

的,程序的执行可能会因为背景值的变化而被分成一些片段。不同的执行片段利用当前的背景状态,执行相应的行为,以适应环境的变化。程序的行为不仅依赖于输入,还依赖于环境,即对于同样的输入,不同的背景,程序执行结果是不一样的。由于背景具有很强干扰性,程序执行过程很难重现。普适计算环境的这些特点使得程序中控制流和数据流变得更加复杂,传统单元测试中的测试用例生成方法对于普适计算软件的测试不再充分。

其次,普适计算环境中的物理背景本质上是不稳定的,有干扰和不精确的,这大大增加了软件测试的难度。例如,定位仪的误差大概在 1m 左右,在测试一个定位软件的时候,对于 1.5m 的误差,就很难区分是软件错误还是硬件故障。

最后,普适计算软件中的各个计算实体(服务或代理)是独立的和自我为中心的,每个实体都只关心与自己相关的环境信息,当一个任务涉及多个计算实体时,这些实体通过它们相应的周围环境相互交互、相互合作。从这个方面来讲,普适计算软件测试与传统的集成测试不同,对于多个程序单元,这里没有明确的调用等级体系。

习题 6.6

什么是普适计算?它具有哪些特点?

6.7 云测试(Cloud Testing)

1. 云测试的概念

云测试是一种软件测试形式,其中网络应用程序使用云计算环境来模拟现实世界的用户流量作为负载和压力测试的一种方式。

云测试本质上是一种软件测试形式,其中测试是利用云端的资源(如主机或服务等),此外,整个测试环境可以按需从云端获得。

我们认为,云测试作为一种云服务和软件测试相结合的产物,尽管现在主要利用云的强大计算能力和存储能力模拟用户的负载来做压力测试,但是随着云计算的不断发展,软件测试中越来越多以往环境中无法解决或需要很大代价来解决的问题都可以利用云解决,与此同时,云测试将会涉及越来越多的测试领域。

2. 云测试的研究背景

当前软件系统的发展非常迅速,软件的规模和复杂度都已经对传统的软件测试提出了很大的挑战。可以总结出以下 3 点:

(1) 对于大规模的软件系统,需要大量的测试用例,即使具有很强计算能力的计算机,也需要花费大量的计算时间,这在敏捷开发中是一个很大的瓶颈(计算能力)。

(2) 软件测试是需要耗费一定资源的,如提供负载能力的模拟资源、一些待测环境的硬件资源等等,在传统的软件测试中,若要满足这些资源是需要很大的代价的,特别是那些硬件资源,因为一个企业不是总具备那些可用的硬件设施的(模拟能力)。

(3) 现在系统对易用性、灵活性和对实时数据的需求也超过了人们的预期,这需要快速的适应不断变化的市场,而传统软件测试的周期已渐渐无法适应这种快速的节奏,对新的快

速便捷的测试的需求也越来越多。

而云计算的出现，为我们解决此类问题提供可行的思路。云是一个可供网络便捷的访问的一个共享的资源池，所需要的资源可以快速方便地建立和配置。由云的概念，我们可以看出，云庞大的资源(包括计算资源)可以为客户提供强大的计算能力，会完成待测系统更大的测试覆盖，那些以前无法完全执行的测试用例云也可以全部通过，而云方便的资源配置管理使得我们使用虚拟技术模拟大量的测试所需资源成为可能。云是以服务的方式为客户提供各种资源，这样系统的开发可以便捷快速地使用网上云提供的测试服务，缩短软件开发周期，从而使得软件系统可以快速地适应市场。因此，在云上应用测试(即云测试)是解决传统软件测试难题的一个很好的方案。

在解决以往问题的同时，云测试还有很多很好的优点，如很好的计价模型——云测试使用“pay-as-you-go”的付费模式，即客户只需为自己消耗的服务资源付费，这种付费模式更加绿色和合理，并且云测试将会驱动测试标准化和提供很好的全球化服务平台。

当然，云测试也是存在很多的问题，如安全、管理以及相关的基础设施的问题。鉴于云测试将是现有系统中的测试问题的很好的解决方案，但是仍然有很多开放性的难题有待解决，因此，有必要对云测试进行更加深入的研究。

3. 云测试当前的研究方向

根据 STITC(Software Testing in the Cloud，云中的软件测试)协会的定义，云测试的研究现集中在以下 3 个方面：

- Test in the Cloud　这是指通过利用云计算基础设施所提供的资源，来促进测试用例在虚拟环境下的并发执行。
- Test of the Cloud　这是指测试那些托管或部属在云端的程序。
- Migrating Testing to the Cloud　这是指将当前的测试过程、测试设备以及它们的基础设施向 Test in the Cloud 或 Test of the Cloud 迁移。以下将逐一介绍它们各自的特点以及现状。

(1) Test in the Cloud

① 原因

现代软件系统的复杂度和规模使得需要更加强大的计算能力和模拟能力才能够全面深入地测试，而云测试则为测试人员提供了几近无限的资源，其中包括计算、存储、网络带宽、便捷的测试服务等等，这些都为上述问题的解决提供了硬件和软件上的支持。

② 优点

云可以模拟测试所需的资源，以前不可能或需要很大的代价才能完成的测试都可以使用云模拟的资源动态地进行测试，并且这些资源是按需供给的，可以方便地建立、扩展和撤销。

云可以提供类似现实的负载能力，使得以往无法做到“真实”的测试系统的性能的问题得以解决。

利用云强大的计算能力可以执行数量巨大的测试用例，那些回归测试以及需要大量测试用例的软件测试都可以很快地在云上执行。

云的计费模型，即现用现付(pay-as-you-go)，使得云测试的收费更加合理，这也是对资

源的一种合理的配置方案。

③ 当前挑战

Ⅰ. 安全问题。不同于本地执行测试,在云中测试的数据可能需要通过网络传输,而如果这是些非常私密的数据,那么会有遭到攻击的风险。

Ⅱ. 标准问题。对在云中测试,我们现在还欠缺统一的标准测试模型和方法,对于适用于云测试的传统应用还需要进行严格的归类以指导之后的工作。

Ⅲ. 管理问题。由于云测试的特性,各种资源需要合理的配置才能发挥云的优势,比如人力资源的分配,必须将测试供应商和需求商协调分配。

④ 当前的方向

Manuel Oriol 和 Faheem Ullah 两人在 *YETI on the Cloud*(YETI 是自动化随机测试工具,可以测试用不同的编程语言实现的程序)这篇论文中讲述了云端版本的 YETI,这个版本的实现是搭建在 Apache 的开源的云框架:Hadoop 之上的,其中的 Map/Reduce 方法使得测试用例能够并行地运行在多台计算结点之上并最终汇总处理,这显著减少了运行时间并提高了测试的执行效率。日本的学者 Toshihiro Hanawa 等人提出来的一个云测试环境——D-Cloud,D-Cloud 侧重于使得测试大规模的分布式并行系统变得可能,并且可以实现配置自动化、利用测试场景的描写来实现故障注入。

(2) Test of the Cloud

① 原因

随着云计算超出人们预料的快速发展,越来越多的企业公司投资云计算的建设,很多云计算的服务和应用都部署起来供用户方便地通过网络访问,大量的文献对云计算的应用和服务进行研究和建模。然而,却只有很少有人去研究如何对它们测试,以及如何衡量云计算服务的质量。

② 优点

对云的测试可以为云计算服务的质量制定指标,使得客户可以放心地在已有的云服务上开发或接受服务。

云计算不断发展需要一个可以保障其发展的对象,而这方面的研究将会促使云的标准化,并且将会影响到测试本身的改革。

③ 当前的挑战

云中的服务不同于传统的软件系统,因为云本身拥有复杂的拓扑结构,部署在其上的应用程序将会有着大规模分布式并行程序所具有的特点,因此,对其测试并不是一件容易的事。

一般而言,对云的测试往往也需要利用云,那么得出的结论是否可信,以什么样的方式来衡量可信度等都是我们所要面对的难题。

④ 当前的现状

在由 Tariq M. King 以及 Annaji Sharma Ganti 两人提出的 *Migrating Autonomic Self-testing to the Cloud* 这篇论文中,他们将 AST(Autonomic Self-Testing,自动化自测试)迁移到云端并结合 TSaaS(Test Software as a Service,测试软件作为一种服务),在虚拟的测试环境中对云的服务进行测试。将 AST 迁移到云端可以实时地监测、分析、执行由于云服务的改变所带来的一系列的可行性的问题。而在 APPLabs 的一篇报告——*Testing the*

Cloud 中，描述了当前日益发展的云计算服务存在的种种问题，包括缺乏控制、安全性、个人隐私问题、数据完整性、可用性及业务接受性等，并提出了对这些问题解决的一些观点，这些观点集中在非功能性测试、功能测试以及相应的测试工具。

(3) Migrating Testing to the Cloud

① 原因

由于云测试的出现，给传统的测试提供了很大的帮助，但是云测试本身仍需要很多的研究才能完全发挥出它的潜力，其中一项就是如何将传统的测试移植到云上。而这个工作包含了从测试的基础硬件设施到软件测试的算法流程的迁移，目前这方面的研究仍然很少。

② 优点

云测试的迁移将会利用传统的软件测试的经验和优点，更加快速地适应云计算时代的节奏，快速推进云计算时代的发展。

③ 当前的挑战

对云测试的迁移还欠缺一些理论的指导，对各种测试方法的迁移的工作量非常巨大。传统的测试如何能利用云的特性从而结合成新的测试需要研究。

④ 当前的现状

在 *When to Migrate Software Testing to the Cloud* 论文中，Tauhida Parveen 和 Scott Tilley 指出并不是所有的问题都适合迁移到云端，他们从两个方向阐述了什么样的软件测试可以迁移到云中：一个是待测程序的特性，如测试用例的独立性、知道该程序操作环境（如依赖的库、组件或其他程序）、还有具备可编程接口等；另一个就是测试类型，如单元测试、大容量自动化测试和性能测试等。*Software Testing as an Online Service: Observations from Practice* 是一篇调查报告，由 Leah Muthoni Riungu、Ossi Taipale、Kari Smolander 等人采访了 11 个企业的经理（6 个供应商、5 个需求商），研究得出将软件测试迁移到云中目前需要考虑到应用、管理、法律以及财务的一些问题。

习题 6.7

1. 查阅有关资料，简要回答什么是云计算。
2. 云测试可以有哪些方式？

6.8 物联网环境下的软件测试（Software Testing in the Context of Internet of Things）

6.8.1 物联网的定义

物联网英文名叫做 Internet of Things，也可叫做 Web of Things。可理解为将现实中的物体与虚拟网络相互连接，形成一个遍及全球的物物相连的网络。物联网的通俗定义是运用射频识别技术、全球定位系统、红外感应器、气体感应器、激光扫描器等各种传感设备或技术，定位现实世界中的物体，采集所需要的各种信息，并且与互联网相连接，形成一个物物相连的巨大网络。物联网发展的目的是最终实现物品与网络的相连，实现统一管理，方便人

类的生产和生活效率。

现实世界中的物体要想与物联网相连接,必须具备一定的条件：首先要有其使用性,并非所有的物体连入网络都有其意义；其次,物体要有信号发送和接收器,并且物体上面应有装置使其具备一定数据处理能力；再次,该物体应遵循相应的协议,在世界网络中有唯一的可被识别的编号；最后,需要具备相应的安全机制,能够保护隐私。

6.8.2　物联网的特征

与互联网相比,物联网具有其自身的特征：

（1）物联网运用了感知技术,例如视频识别技术。物联网需要在其网络中部署大量的感应结点,并部署大量的感应器,用来接收和获取物体发出的各种不同频率的信号。传感器在获得数据的同时,还要不停地更新自己采集信息的频率,以动态地不断获得物体信息的变化。

（2）物联网仍然是互联网的基础上建立的一个网络。其技术基础和框架体系仍是互联网。物联网在互联网的数据传输基础上融入了传感技术,实现了对物体的相连。传感器搜集到的信息需要通过网络传输,这一数据量是十分巨大的。在传输过程之中,必须要像互联网一样,满足数据传输的准确和快速性的要求。因此,物联网的网络体系也必须要适应不同的网络条件和协议体系。

（3）物联网结点不仅能够运用传感技术识别物体,其本身也是一种智能处理设备,能够实现对物体的智能控制,并将传感技术与其他智能技术如云计算技术、模式识别相结合,不断扩宽物联网的应用领域。物联网在获取海量数据信息的同时,要根据不同用户的不同需要,不断丰富自己的应用范围。

6.8.3　物联网的体系结构

目前普遍认可的物联网体系结构分为 3 层：应用层、网络层和感知层(如图 6-10 所示)。



图 6-10　物联网体系结构

感知层由各种结点及其相应的感知设备所组成。相应的感知设备包括 RFID(射频识别)读写器、RFID 标签、温度湿度感应器、浓度探测器、摄像头等。感知层是物联网的最底层结构,类似于互联网的物理层。感知层的作用是感知物体,采集物体信息。

网络层又可称为网络通信层，它将感知层所获得的数据通过无线传输方式连入到物联网的网络之中，根据不同的需要送抵不同的终端来处理。

应用层是物联网提供给用户直接使用的接口，应用层需要结合不同行业和不用用户的需求，筛选和处理网络层传输的信息，提供给用户使用。应用层是物联网发展的最终目的。

6.8.4 物联网的核心技术

物联网的 4 大核心技术包括 RFID 技术、传感网技术、M2M(Machine to Machine，机器到机器)技术和两化融合技术。

1. RFID 技术

RFID 射频识别是一种非接触式的自动识别技术，它通过射频信号自动识别目标对象并获取相关数据，识别工作无须人工干预，可工作于各种恶劣环境下。RFID 技术可识别高速运动物体并可同时识别多个标签，操作快捷方便。RFID 是一种简单的无线系统，只有两个基本器件，该系统用于控制、检测和跟踪物体。系统由一个阅读器和很多标签组成。

2. 传感网技术

传感网的定义为：随机分布，集成有传感器、数据处理单元和通信单元的微小结点，通过自组织的方式构成的无线网络。

借助于结点中内置的传感器测量周边环境中的热、红外、声呐、雷达和地震波信号，从而探测包括温度、湿度、噪声、光强度、压力、土壤成分、移动物体的大小、速度和方向等物质现象。

3. M2M 技术

M2M 是 Machine to Machine/Man 的简称，是一种以机器终端智能交互为核心的、网络化的应用与服务。它通过在机器内部嵌入无线通信模块，以无线通信等为接入手段，为客户提供综合的信息化解决方案，以满足客户对监控、指挥调度、数据采集和测量等方面的信息化需求。M2M 根据其应用服务对象可以分为个人、家庭、行业三大类。

4. 两化融合技术

两化融合是信息化和工业化的高层次的深度结合，是指以信息化带动工业化、以工业化促进信息化，走新型工业化道路；两化融合的核心就是信息化支撑，追求可持续发展模式。M2M/物联网技术是两化融合的补充和提升，两化融合也是物联网 4 大技术的组成部分和应用领域之一。

6.8.5 物联网的安全特性

1. 哪些特性决定了物联网的安全要求

物联网的特性决定了物联网所采取的安全要求。物联网有着传感网的基本特性和自身的一些高级安全特性。

基本特性如下：

（1）流动性。物联网上的设备通常都是移动的，在不同的地方可能会使用不同的接入商接入网络。

（2）无线性。物联网设备间采用各种无线协议进行传输，例如，Bluetooth、802.11、WiMAX、Zigbee、GSM/UMTS。这些无线传输信号易被附近的攻击者截获。

（3）嵌入式应用。一些物联网设备通常只有单独的作用，被嵌入进某一系统中，因此需要针对于此，采用独特的发现模式。

（4）多样性。未来接入物联网的设备将从复杂的计算机到最简单的RFID标签，最简单的设备其隐私也需要得到保护。

（5）规模性。物联网得到普及之后，接入物联网的设备将越来越多，越来越多的东西嵌入到日常生活中来，这将加大隐私保护难度。

高级安全特性如下：

（1）用户识别。用户在使用系统前必须首先经过验证才能使用。

（2）安全存储。必须要保证存储在物联网中的敏感信息的保密性和完整性。

（3）身份管理。通过一定的认证机制来限制不同用户的使用权限。

（4）安全数据通信。安全的数据通信包括很多内容，需要通信前首先验证通信双方，确保数据的保密性和完整性，同时部分情况下，需要防止否认交易，还要保护通信实体的身份。

（5）可用性。未经授权的人或系统无法拒绝授权用户的访问。

（6）安全的网络访问。物联网上的设备只有授权才能连入网络或访问其他设备。

（7）安全的执行环境。物联网需要一个安全的、高效运作的编码机制来防止数据偏差。

（8）防篡改。防篡改指的是这些设备即使落入攻击者手中或者受到物理攻击，需要采取一定的机制来继续维持这些安全要求。

2. 无线传感网络的安全特性

物联网是互联网的拓展，具有互联网本身的安全特性，这里不着重论述。物联网也是无线传感网络的延伸，安全方面也具有无线传感网络的安全特点。

无线传感网络的安全特点如下：

（1）独立结点的资源有限。无线传感网络的结点通常为一个很小的设备，这一很小的设备需要集成处理器、内存、感应器、电源、安全保护设备。因此它无法进行高复杂度的加解密计算。需要设计高效的加解密算法来保证独立结点数据传输的安全。同时，单个结点的数据存储能力有限，无法长期保存数据，出于安全性的要求，也不应该长期保存数据。

（2）结点易受物理攻击。无线传感网络的结点总是部署在无人值守的环境中，结点很容易受到攻击，受到攻击后易失效，并且难以维护。甚至在受到物理上的攻击后，结点很可能丧失作用。在军事应用中，这种无线传感结点更易遭到毁灭性的攻击，而造成巨大损失。

（3）传输介质的复杂性和不稳定性。无线传感网络中信号传输的无线介质易受周围环境，例如恶劣天气的干扰，使数据传输的差错加大，结点附近各种信道传输数据也易发生差错，一些攻击者可能会伪造一些结点来实施攻击。

（4）无线传感网络没有基础性架构。无线传感网络中没有专用的信号传输设备，各个结点相互配合传输数据，这使得很多在有线互联网中已经使用的安全架构无法在无线传感

网络中部署。所以说有线网中已经成熟的安全架构要在适应无线传感网络特性的基础上结合使用并作出相应改变。

6.8.6 物联网安全问题分类

物联网体系分为感知层、网络层、应用层。其中网络层与应用层的安全问题与传统的互联网安全问题相类似,这里不再详细讨论。物联网除具有无线传感网络的安全问题特点外,物联网安全问题主要围绕其感知层的技术,尤其是RFID射频识别技术的安全问题展开。其安全特点可分为6个方面:物理安全、传输安全、隐私问题、攻击手段、物联网平台安全问题、物联网网络层和应用层安全问题。

1. 物理安全

(1) 设备的物理安全。物联网所连接的是现实世界中的物体,这些物理部件部署在无人值守的场合,攻击者可以轻易接触到这些物体,对他们造成物理上的破坏甚至直接盗走物理设备,造成经济上的损失。嵌入RFID标签的物体若RFID标签受到物理攻击,会直接损坏RFID标签,RFID读写器便无法获得该物品的射频信号,从而使该物体从物联网上"失踪"。

(2) 结点的物理安全。物联网中部署的各式数据获取和数据传输的无线传感网络结点很多也是部署在无人的环境中,它们也易受到物理上的攻击。同时,物联网中的独立结点功能并不是非常强大,它们的电池能量少,安全保护能力薄弱。结点在不停地与外界环境进行数据感知以及监控附近环境设备状态的同时,它们本身也极易成为攻击的对象。结点受到攻击,将直接影响其对周围设备的感知能力。

2. 传输安全

(1) RFID读写器和RFID标签之间数据传输的安全。RFID标签有被动式和主动式两类,这两类标签均需要与RFID读写器之间进行数据通信。RFID标签与RFID读写器之间通过无线信号进行传输,在传输过程中,这些数据传输信号几乎是直接"暴露"在空气中的,很容易被攻击者截获,所以需要有相应的保护措施。在物联网时代,如何在数以亿计的RFID标签和读写器之间建立一个可靠的传输环境,也是RFID技术运用在物联网上遇到的一个重大难题。

(2) 物联网感知层结点数据传输的安全。物联网感知层结点传输的数据因不同的环境、不同的设备、不同的通信标准各异,故结点传输的数据结构非常复杂,数据量非常大。感知网络应用丰富多彩,从导航到温度监控、从水文控制到自动化,感知网络的数据传输没有统一标准,故无法拥有一个有效的安全保护架构。结点的数据更新频率非常高,这些数据具有很强的实时性特征,又是多源异构性数据。与传统的互联网安全技术相比,物联网时代的网络数据传输将面对更加复杂的数据结构,更加困难的数据防御攻击要求。

(3) 物联网主干网络的传输安全。物联网中各类RFID读写器获取的数据,各类结点的数据最终都需要传输到主干网络上,再根据不同的应用需要传送到不同的地方。在未来,将有数以亿计的物联网设备,大量设备集群出现,一旦网络出现故障,海量的数据传输将会使网络出现严重的拥塞。虽然根据设计,物联网的主干网络拥有较强的安全保护能力,但是

随着未来接入物联网的设备越来越多,大量的拥塞数据充塞在物联网主干网络上还是会带来拒绝服务攻击。物联网主干网络现在的安全体系仍是基本沿用的互联网时代的安全体系,尚未提出一个健全的完全适用于“物体”之间通信的安全体系。

3. 隐私问题

隐私问题是物联网时代的一个重大安全问题,其重要性远远超过互联网时代隐私问题的重要性。物联网的网络层和应用层除具有传统互联网的隐私问题外,其和现实世界中物体相接触的感知层带来了新的安全隐私问题,具体来讲就是 RFID 标签带来的隐私问题。物联网是物物相连的网络,很多连入物联网的物体和我们日常生活息息相关。例如,一些日常生活使用的物体中嵌入的 RFID 标签往往含有我们的个人信息,一旦不法分子得到这些标签,也就意味着有可能通过解密措施获得人们的隐私信息,甚至,如果攻击者掌握了标签如何追踪,就可以通过追踪这些标签随时随地知道使用者的位置,这会给用户带来重大的安全隐患。

4. 攻击手段

物联网是传统互联网的延伸,物联网也会受到类似传统互联网遭到的攻击。

物联网主要受到的攻击包括数据驱动攻击、假冒攻击、恶意代码攻击、拒绝服务攻击。

(1) 数据驱动攻击是指通过向某个程序或应用发送数据,以产生非预期结果的攻击,通常为攻击者提供访问目标系统的权限。数据驱动攻击分为缓冲区溢出攻击、格式化字符串攻击、输入验证攻击、同步漏洞攻击、信任漏洞攻击等。通常向传感网络中的汇聚结点实施缓冲区溢出攻击是非常容易的。

(2) 假冒攻击是一种主动进行的攻击手段,物联网无线传感网络的 RFID 读写器、RFID 标签都是攻击者可以随时能接触到的,各无线传感结点也是独立部署的。攻击者可以伪造这些标签、伪造结点向物联网中的其他设备发出信息,这一攻击手段对物联网设备的相互信任和各结点间的协同工作造成了很大的威胁。

(3) 恶意代码攻击是指恶意程序通过某个漏洞入侵网络,相当于互联网时代的可以在网络传播的计算机病毒。在物联网时代,由于无线传感网络的环境和物物相连的特性,恶意代码一旦找到入侵入口,进入网络中后,它会通过便捷的网络环境迅速向整个网络中扩散。

(4) 拒绝服务攻击是物联网受到的攻击手段中最重要的一种,后面将对这一攻击进行详细介绍。未来,数以亿计的物联网装置,大量设备集群出现,一旦网络出现故障,海量的数据传输将会使网络出现严重的拥塞,产生拒绝服务攻击,这种攻击手段一般出现在感知层与主干网络中。

5. 物联网平台安全问题

在未来,物联网得到普及之后,巨大的物联网平台上必然存在着各式各样的物联网应用。学校、工厂、商店、超市内部都具备各自的物联网系统。这些各自独立的物连网最终需要统一相联,这就需要建立一个统一的物联网应用层安全技术标准来管理这些不同物联网应用。否则,物联网应用层安全平台必然无法适应未来海量的物联网应用。但这样一来,如何统一管理物联网各种不同应用层设备间的安全日志等安全信息将会成为新的物联网应用

层的安全管理问题。如果不能建立一个协调统一的机制，将会严重削弱物联网应用层安全平台和各应用平台间的相互信任关系。

6. 物联网网络层和应用层的安全问题

物联网网络层和应用层仍具有传统互联网面临的所有安全问题。并且，由于物联网上的结点更加丰富，采集的数据格式更加多样，传输的条件更加复杂，因此物联网网络层和应用层所面对的传统安全问题更加复杂。

6.8.7 物联网安全的对策

互联网在应用之前没有考虑到自身的安全问题，随着互联网的迅猛发展，它带来的安全问题也越来越严重。黑客、病毒、网络欺诈、隐私泄露，每年都给全世界造成了无可计数的经济损失，给互联网在全球的运用蒙上了一层阴影。然而，互联网所涉及的毕竟还是虚拟数据，而物联网所连接的都是现实世界中的物体，一旦出现重大安全问题，其后果是难以想象的。因而，我们需要在物联网正式投入使用前，首先解决或者避免这些安全问题的发生。

物联网继承了互联网的一些特性，应该沿用互联网的大部分安全机制，并在此基础上针对自己的特点进行补充和调整。传统网络的 OSI 七层结构相应的安全措施并不能完全适用于物联网，物联网需要结合自己感知层的特性提出自己相应的安全对策。

其实想要充分解决物联网的安全问题，不仅是科技层面的技术问题，还是国家层面的法律问题和公民层面的道德问题，只有把技术、法律、道德三者相结合，才能从根本上解决物联网的安全问题。以下从技术上提出一些解决物联网问题的对策。

1. 物理设备处理的对策

嵌入 RFID 标签的物体若被盗，RFID 读写器定期发出信号，若长久接收不到 RFID 标签返回的信息，RFID 读写器应向结点发出“说明”该结点范围内的某一物品已经“丢失”，该结点向一定范围的其他结点发出同样“说明”，这个区域的 RFID 读写器便不再接收这一个 RFID 标签发出的射频信号。同时，一些政府、军事使用的涉及隐私、机密的 RFID 标签若长期接收不到 RFID 读写器的请求，这时就有可能被盗，其自身也要一定的自我销毁能力，即使标签落入不法分子手中，其存储的内容也不会被获得。

2. 感知层 RFID 标签的安全对策

(1) 杀死标签。带有 RFID 的标签的某些物体在某些运用后，若使用者不想再使用该 RFID 标签，可以通过杀死该标签，杀死该标签的缺点是标签被摧毁后无法再被利用，造成成本上的浪费，这一点会影响物联网的普及。同时，杀死标签会影响其反向的追踪，例如退货。

(2) 干扰标签和读写器之间的射频传输。干扰标签和读写器之间数据传输主要有 3 种方法，即改变频率、主动干扰和静电屏蔽。

改变频率是指使用者可以通过改变 RFID 读写器或者 RFID 标签的频率，使得没有权限的用户无法探测到读写器和标签之间的通信，这一方法的缺点是需要大大增强 RFID 标签和 RFID 读写器之间的功能，使它们的成本更贵。使用者还可以通过主动干扰的方法，用

一个能发出无线电信号的设备，干扰附近 RFID 读写器的工作，以保护 RFID 标签，但是这有可能影响其他物联网设备的正常使用。静电屏蔽指的是将 RFID 标签置于一个用金属网或者金属薄片制成的外罩中，保护标签使 RFID 读写器无法对其进行读写操作，这一方法是某些场合有其独特的应用，但是缺点是造成成本的上升和使用的不便。

(3) RFID 读写器和 RFID 标签数据传输间的加密和认证措施。采用加密或认证手段确保 RFID 阅读器和 RFID 标签之间的数据安全。例如，Hash 锁、Hash Lock 协议、随机化 Hash Lock 协议、带别名的双向认证、基于杂凑的 ID 变化协议、分布式 RFID 询问-相应认证协议、三方认证等措施来保护读写器和标签数据传输的安全，但是这要综合考虑加密算法的复杂程序和标签的成本及计算能力。在未来，物联网在构建 RFID 射频系统时要根据实际情况考虑是否采用有加密或认证功能的系统。

3. 物联网感知层结点的安全对策

在前面分析物联网的安全问题时，已经分析了物联网感知层结点包括 RFID 读写器安全问题。目前已经提出两大类安全对策：

第一种对策是采取传统的加密认证机制和访问控制技术。认证机制是通信双方在通信前确认发送者和接受者的身份，加密就是在数据传输过程中用密钥通过加密和解码来确保数据传输的机密性。访问控制技术在物联网时代，就是给人和物体授权，限制他们访问某些物体的权限和不具备的权限。由于物联网中更多的是连入网络的物体，访问控制技术将会变得更加复杂。

第二种对策是在感知层传输数据时构建专用的通信协议和通道。通过构建专用的通信协议和通道确实能在感知层传输数据时起到非常好的安全效果。但是，它在带来传输高性能的同时增加了资金的投入，降低了物联网不同应用层之间的数据共享能力。

4. 拒绝服务攻击

拒绝服务攻击有两种方式，一是拒绝服务攻击(DoS)，二是分布式拒绝服务攻击(DDoS)。

(1) 拒绝服务攻击。拒绝服务攻击是指一个用户占用大量的共享资源，使系统没有多余的资源给其他用户使用的攻击方法。攻击者发动拒绝服务，首先向服务器发送众多的带有虚假地址的请求，服务器发送回复信息后等待回传信息，由于地址是伪造的，所以服务器一直等不到回传的消息，分配给这次请求的资源就始终没有被释放。当服务器等待一定的时间后，连接会因超时而被切断，攻击者会再度传送新一批请求，在这种反复发送伪地址请求的情况下，服务器资源最终会被耗尽(如图 6-11 所示)。

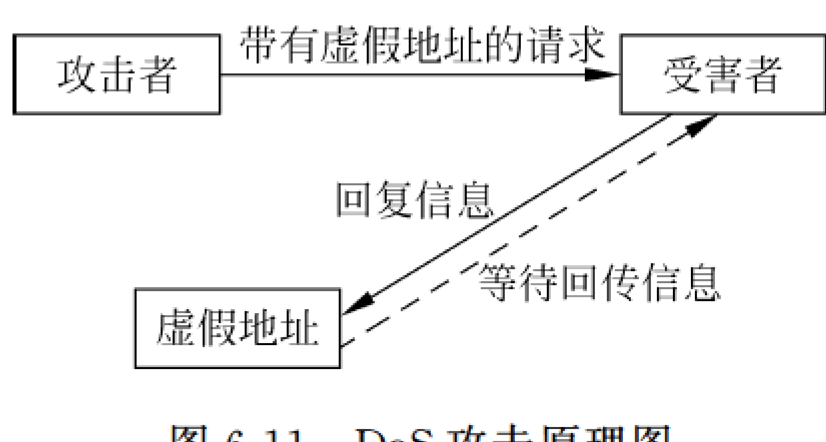


图 6-11　DoS 攻击原理图

(2) 分布式拒绝服务攻击。分布式拒绝服务攻击(DDoS)是一种基于 DoS 的特殊形式的拒绝服务攻击，是一种分布、协作的大规模攻击方式，主要针对比较大的站点，像商业公司、搜索引擎和政府部门的站点。DDoS 攻击是利用一批受控制的机器向一台机器发起攻击、这样来势迅猛的攻击令人难以防备，因此具有较大的破坏性。探测扫描大量主机找到可

以入侵的目标主机，通过一些远程漏洞攻击程序，入侵有安全漏洞的目标主机并获取系统的控制权，在被入侵的主机上安装并运行 DDoS 分布端攻击守护进程，然后利用多台已被攻击者控制的机器对另一台单机机型扫描和攻击，在悬殊的力量对比之下，使被攻击者很快失去反应能力。整个过程都是自动化的，攻击者可以在几秒钟内入侵一台主机并安装攻击工具，这样在一小时内就可以入侵数千台主机。

DDoS 攻击分为攻击者、主控端、代理端三层。攻击者操纵整个攻击过程，它向主控端发送攻击命令。主控端接受攻击者发来的攻击命令，并且可以把这些命令发送到代理主机上。代理端运行攻击器程序，接受和运行主控端发来的命令。代理端主机是攻击的执行者，真正向受害者主机发送攻击(如图 6-12 所示)。

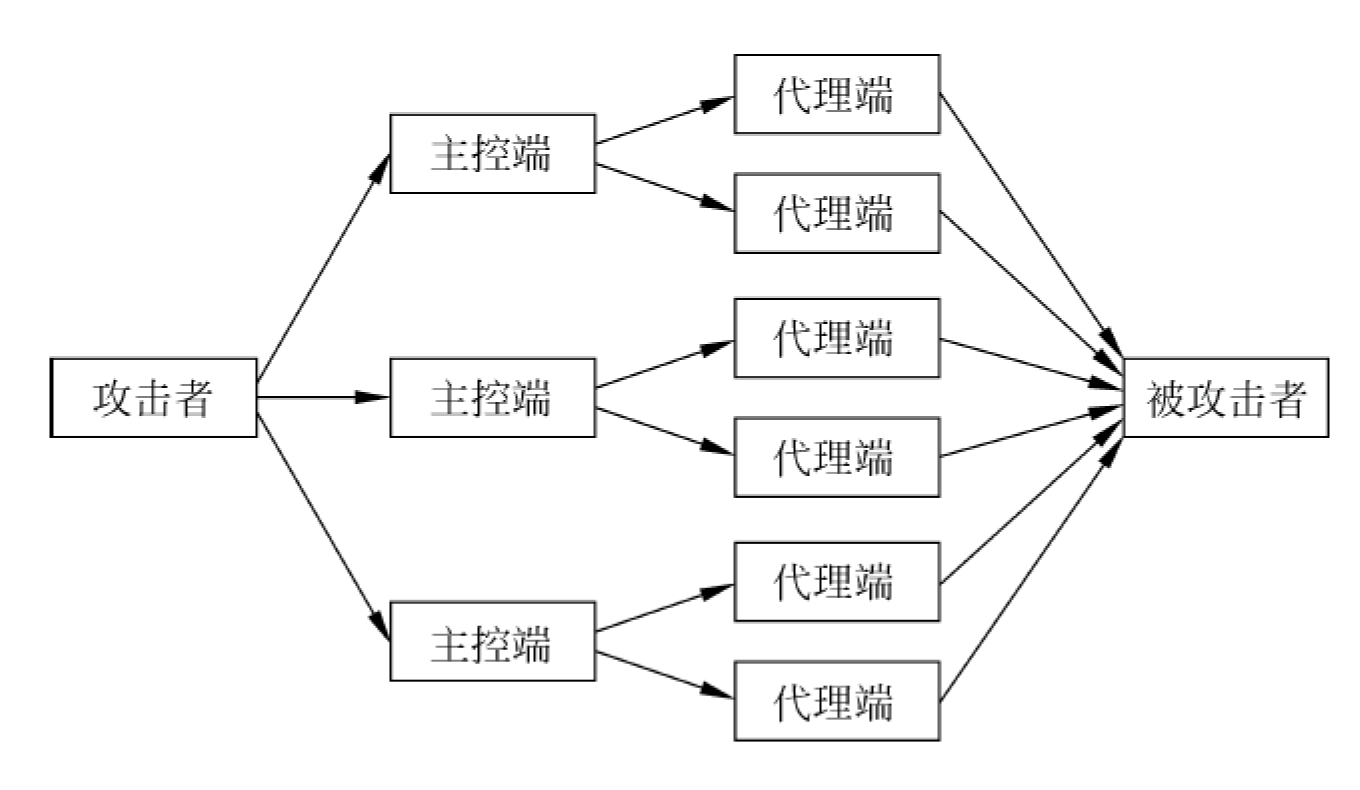


图 6-12　DDoS 攻击原理图

(3) 拒绝服务攻击的对策。传统的解决拒绝服务攻击和分布式拒绝服务攻击的方法有黑洞法、设置路由访问控制列表过滤和串联防火墙安全设备等几种方法，这些方法的效率均不高。在物联网时代，想要解决拒绝服务攻击，关键在于智能化。

智能化 DDoS 防护系统由检测器和防护器两部分组成。该系统使用便捷，部署简便，能在不改变原来网络架构的基础上，实行动态、智能化防御。防护器采用并联方式安装在主干网络上，当检测器检测到网络中有不良流量进行攻击时，检测器向防护器发出警报。DDoS 防护器知道攻击发出的地址、攻击的目的地后立刻开始工作，通知路由器，路由器随后将对攻击目标发送的不良数据全部发送到防护器端，防护器对这些数据进行分析和验证，不良流量将被截获丢弃，而正常的数据流量将通过路由器传送到目的地址，不影响它们的传输。

习题 6.8

1. 什么叫物联网？物联网具有哪些安全性问题？
2. 物联网安全性测试中应该包含哪些方面？

6.9　并行软件测试(Concurrent Software Testing)

并行程序通常包含两个或多个并发执行的线程，这些线程通过相互合作来完成某些任务。使用多线程技术可提高计算效率，同时由于一些问题本身具有并发的特性，采用多线程

技术解决这类问题也更加自然。目前,Java、C#和Ada等语言对并发程序设计都提供了直接支持,而C和C++语言通过一些多线程库(如POSIX Pthread库)间接支持并发程序设计。随着多核技术的发展和实际应用需求的增加,并发程序的开发应用变得越来越广泛。

并行程序的目标是提高软件性能,充分利用可用资源。一个并行软件由若干相互合作的进程共同完成一个任务,在很多应用领域中可以大大减少计算时间,例如,面向服务的软件系统、天气预报系统、动力学分子模拟、生物信息和图像处理。尽管硬件技术飞速发展,速度和性能不断提升,但这种进步不可能是无止境,要进一步提高软件在处理大规模数据时的运行速度和工作效率,还要在改进软件结构、充分利用软硬件资源方面着手。因此,人们对于并行软件的需求越来越高,并行软件测试也因此显得越来越重要。

对于传统的软件,很多测试问题都可归结为一些测试准则以及相应的支持工具。所谓测试准则就是测试用例集合应该满足的标准,它可以作为测试用例生成的向导,例如,结构化测试准则就是利用代码、程序实现和结构方面的信息生成测试用例,以达到相应的代码覆盖、变量的定义引用覆盖和其他覆盖标准等。测试准则可以保证在不降低软件错误检测能力的情形下,允许人们选择尽可能少的测试用例。同时测试准则使得测试更加系统化,并使得测试活动可以度量。并行软件测试一方面可以考虑将传统的软件测试方法应用于测试并行软件,但并行软件与传统的软件毕竟有着很大不同,最主要方面包括:并行程序间通讯、同步和执行的非确定性。这些特性使得并行软件测试更为复杂,因为并行软件执行过程中涉及大量的交互、极大数量的执行路径,因此,并行软件测试既要考虑顺序特性,又要考虑并行特性。

随着服务计算、云计算等新兴技术的发展,并行软件已经成为这些新技术的共同基础,并行软件测试是保证这些新型应用质量和可靠性的关键技术,因此研究并行软件的测试机制和故障模式具有非常重要的意义。

并行进程之间的交互有两种形式,一种是消息传递,一种是存储共享。两种方式下都需要通过信号量或监控进行同步,这就形成并行软件的几个特征,即非确定性、同步和通信。根据这些特性,人们已经提出了一些并行测试的挑战:开发并行程序的静态分析方法、在非确定性程序中检测无意的临界资源和死锁条件、在非确定性存在的情况下执行某条指定的路径、使用同样的输入重现某次测试过程、产生非确定性程序的控制流图、提供一个将顺序程序测试准则应用于并行程序的测试框架作为理论基础,并研究其可行性,研究并定义并行程序的基于控制流和数据流的测试覆盖准则。

并行程序测试研究最早可以追溯到20世纪70年代。与顺序程序类似,并发程序测试可分为基于规格说明的功能性测试和基于程序实现的结构性测试,即黑盒测试和白盒测试。黑盒测试方法根据软件规格说明构造状态机模型,选择高层事件执行序列进行测试。白盒测试方法基于程序实现,通过分析程序源代码,选择程序语言的同步序列进行测试。基于程序实现的测试方法主要可分为静态分析方法和动态分析方法,静态分析直接分析并发程序源代码,不使用任何运行时的信息,精度较低;动态分析方法分析程序运行时的信息,精度较高。静态分析方法采用的并发程序模型主要有扩展控制流图法和可达图,动态分析方法主要有可达性测试方法等。

习题 6.9

结合并行程序设计经验，总结并行程序测试与普通程序测试的不同。

6.10 嵌入式软件测试（Embedded Software Testing）

IEEE 对于嵌入式系统的定义是：嵌入式系统是用于控制、监视或者辅助操作机器和设备的装置（Device Used to Control, Monitor, or Assist the Operation of Equipment, Machinery or Plants）。从这个定义中可以看出，嵌入式系统是软件和硬件的综合体，可以涵盖机电等附属装置。目前国内普遍认同的定义是：嵌入式系统是以应用为中心、以计算机技术为基础，软硬件可以剪裁，适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统。

嵌入式系统对可靠性的要求比较高。嵌入式系统安全性的失效可能会导致灾难性的后果，即使是非安全性系统，由于大批量生产也会导致严重的经济损失。这就要求对嵌入式系统，包括嵌入式软件进行严格的测试、确认和验证。随着越来越多的领域使用软件和微处理器控制各种嵌入式设备，对日益复杂的嵌入式软件进行快速有效的测试愈加显得重要。

嵌入式软件测试/嵌入式测试或叫交叉测试（Cross-test）的目的与非嵌入式软件是相同的。但是，在嵌入式系统设计中，软件正越来越多地取代硬件，以降低系统的成本，获得更大的灵活性，这就需要使用更好的测试方法和工具进行嵌入式和实时软件的测试。

嵌入式软件测试作为一种特殊的软件测试，它的目的和原则同普通的软件测试是相同的，同样是为验证软件质量以及是否达到软件可靠性要求而对软件进行测试，但嵌入式软件有其自身特点：

（1）嵌入式软件是在特定的环境下才能运行的软件，因此，嵌入式软件测试最重要的目的是保证嵌入式软件能在特定的环境下更可靠地运行。

（2）嵌入式软件测试不仅要保证其在特定的环境下运行的可靠性，还要保证其实时性，例如在工业控制中，在某些特定环境中如果不具备实时响应的能力，就可能造成巨大的损失。

（3）嵌入式软件产品为了满足高可靠性的要求，不允许内存在运行时有泄漏等情况发生，因此，嵌入式软件测试除了对软件进行性能测试、GUI 测试、覆盖分析测试等，还需要对内存进行测试。

（4）嵌入式软件在完成和硬件的集成测试之后，还要对相应的嵌入式产品进行测试，以保证嵌入式产品除了能够满足所有功能的同时可以安全可靠地运行。

基于以上特点，嵌入式软件测试需要遵循以下一些特殊的原则：

（1）嵌入式软件测试对软件在硬件平台的测试是必不可少的。

（2）嵌入式软件测试需要在特定环境下对其进行测试，例如，对某些软件在工业强磁场的干扰下测试，以保证嵌入式软件可靠性。

（3）必要的可靠性负载测试，例如，测试某些嵌入式系统能否连续 1000 个小时不断地

工作。

(4) 除了进行功能测试,还需要对实时性进行测试,在判断系统是否失效方面,除了看它的输出结果是否正确,还应考虑其是否可以在规定的时间里输出了结果。

(5) 需要在特定的硬件平台上对嵌入式软件进行性能测试、内存测试、GUI 测试、覆盖分析测试,这些测试都可以利用相关工具进行。

(6) 需要对嵌入式产品进行测试。

在嵌入式系统中,内存问题危害很大,不容易排查,主要有内存泄露、内存碎片和内存崩溃 3 种类型。对于内存问题态度必须要明确,那就是早发现早修正。其中内存泄露主要由于不断分配的内存无法及时地被释放,久而久之,系统的内存耗尽。即使细心的编程老手有时后也会遭遇内存泄露问题。有测试过内存泄露的朋友估计都有深刻地体验,那就是内存泄露问题一般隐藏很深,很难通过代码阅读来发现。有些内存泄露甚至可能出现在库当中。有可能这本身是库中的错误,也有可能是因为程序员没有正确理解它们的接口说明文档造成错用。

内存碎片比内存泄露隐藏还要深。随着内存的不断分配并释放,大块内存不断分解为小块内存,从而形成碎片,久而久之,当需要申请大块内存时,有可能就会失败。如果系统内存够大,那么坚持的时间会长一些,但最终还是逃不出分配失败的厄运。在使用动态分配的系统中,内存碎片经常发生。目前,解决这个问题最有效的方法就是使用工具通过显示系统中内存的使用情况来发现谁是导致内存碎片的罪魁祸首,然后改进相应的部分。

内存崩溃是内存使用最严重的结果,主要原因有数组访问越界、写已经释放的内存、指针计算错误、访问堆栈地址越界等等。这种内存崩溃造成系统故障是随机的,而且很难查找,目前提供用于排查的工具也很少。

一般来说,嵌入式软件测试在 4 个阶段上进行,即模块测试、集成测试、系统测试、硬件/软件集成测试。前 3 个阶段适用于任何软件的测试,硬件/软件集成测试阶段是嵌入式软件所特有的,目的是验证嵌入式软件与其所控制的硬件设备能否正确地交互。嵌入式软件测试工具包括 GUI 测试工具、覆盖分析工具、性能分析工具、内存分析工具等。

习题 6.10

1. 手机是一种嵌入式系统,简述读者自己平时在选购手机时做过的测试。
2. 列举读者自己身边的嵌入式系统。

6.11　高可信软件测试(High Confidence Software Testing)

随着软件在人们生活、学习和工作中的日益渗透和普及,由于软件的缺陷、漏洞、故障和失效而给人们生活、生产带来不便和重大损失的事件越来越多,软件的可信性问题日益突出,并已经成为国际上普遍关注的问题。

软件的可信性体现的是人们对软件性能的一种新的抽象追求,它其实也包括了正确性、可靠性、安全性、可用性、可控性、完整性、实时性和可预期性等许多传统特性。目前关于可信性还未形成比较完整和有说服力的解释,但人们已经普遍认为软件的可信性应该是软件

的行为及结果符合人们的预期，在操作失误、环境影响、外部攻击等干扰时，仍然可以提供连续的服务。

高可信软件技术是当前软件技术面临的重大挑战，从科学意义上，它要求人们对软件系统开发和运行等规律有更深一步的认识；从应用价值上，它关系到人们在信息社会对信息基础设施的依赖和可信程度的提高。

高可信软件工程涉及一系列科学问题：第一、软件系统的行为特征，如何定性/定量地描述软件的行为？如何建立各类复杂的软件结构和系统对应的系统行为？这是软件理论的基本问题，软件的静态语法与其动态语义的分离是造成软件行为难于描述和推理的原因。随着软件规模的增大，软件中并发、实时、分布、移动等特性的出现，这些问题的认识亟待深入；第二、软件可信性质与软件行为的关系，如何描述软件可信性质及其与软件行为的关系？我们看到，上述关于软件可信性质的描述是非形式化的抽象陈述，必须建立起性质和软件行为之间的内在联系及其严格的描述，才能在软件开发中，设计并验证所需的可信性质；第三、面向软件可信性质的设计和推理，如何将软件可信性质（通常非操作性的）融入软件设计（操作性）？可信性质通常是软件系统的全局约束，伴随着软件开发过程逐步地“设计”出来，最终获得这些可信性质。如何针对可信性质发现一种分而治之的策略从而控制复杂性，是进行面向可信性质的软件设计和验证的关键；第四、软件系统的可信性质的确认，如何发现和评估软件系统是否具有可信性质？量化是一种工程科学成熟的重要标志，需要对软件可信性质有合适的度量方法，并能在软件过程中进行跟踪。

软件系统的可信性质是指该系统需要满足的关键性质，当软件一旦违背这些关键性质时就会造成不可容忍的损失，称这些关键性质为高可信性质。由此进一步提出了可信软件的定义。可信软件是指软件系统的运行行为及其结果总是符合人们的预期结果，在受干扰的环境下仍能提供连续服务。广义地讲，软件可信性包含的特征属性有可用性（Availability）、可靠性（Reliability）、安全性（Security）、可维护（Maintainability）、完整性（Integrity）等。因此，高可信软件测试主要围绕这些方面，利用前面介绍的各种测试方法和技术系统地进行。

习题 6.11

1. 高可信软件应该包含哪些特性？
2. 人们为什么在高质量软件、高可靠性软件等名称之后，提出高可信软件的概念？

6.12 网构软件测试（Internetware Testing）

Internet 的出现和普及使计算机软件开发、部署、运行和维护的环境开始从封闭、静态、可控逐步走向开放、动态、难控。软件系统需要适应 Internet 发展的要求，具备开放的结构，拥有动态协同、在线演化、环境感知和自主适应的能力，对于这种新的软件形态，北京大学的杨芙清院士、梅宏院士和南京大学的吕建教授等国内学者提出了网构软件的概念，十多年来，他们带领自己的研究组围绕网构软件展开了深入系统的研究，已经积累了一整套的理论、方法和实践经验。网构软件系统中的实体元素被组织为分布、自治、异构的构件，具有独

立性、主动性和自适应性,这给网构软件的测试带来了新的挑战。

网构软件是在 Internet 开放动态和多变环境下软件系统基本形态的一种抽象,它既是传统软件结构的自然延伸,又具有区别于在集中封闭环境下发展起来的传统软件形态的独有的基本特征,它们包括:

(1) 自主性。网构软件系统中的软件实体具有相对独立性、主动性和自适应性。自主性使其区别于传统软件系统中软件实体的依赖性和被动性。

(2) 协同性。网构软件系统中软件实体与软件实体之间可按多种静态连接和动态合作方式在开放的网络环境下加以互连互通、协作和联盟。协同性使其区别于传统软件系统在封闭集中环境下单一静态的连接模式。

(3) 反应性。指网构软件具有感知外部运行和使用环境并对系统演化提供有用信息的能力,反应性使网构软件系统具备了适应 Internet 开放动态和多变环境的感知能力。

(4) 演化性。网构软件结构可根据应用需求和网络环境变化而发生动态演化,主要表现在其实体元素数目的可变性、结构关系的可调节性和结构形态的动态可配置性。演化性使网构软件系统具备了适应 Internet 开放动态和多变环境的应变能力。

(5) 多态性。网构软件系统的效果体现出相容的多目标性,它可根据某些基本协同原则在动态变化的网络环境下满足多种相容的目标形态。多态性使网构软件系统在网络环境下具备了一定的柔性和满足个性化需求的能力。

由于网构软件的生成元素、生成过程的特征及其所面临环境的开放、动态和多变性,因此决定了网构软件的测试必须要有新的技术、方法来适应它的发展。例如,对传统软件系统正确性的概念和相应的质量标准是确定的,正确性的验证与质量测试和评价均依此而定。然而在 Internet 环境下一个软件系统所能达到的目标不仅取决于系统的输入而且还取决于开放和动态变化的外部环境,因此,一个软件系统所能达到的目标难以用传统的正确性概念加以概括,也就是说,网构软件系统是一个具有多个相容目标的系统,其正确性可能表现为传统正确性描述的一个补充,同时,多目标的正确性概念也是网构软件演化的基本依据,正确性概念的变化可能会导致传统软件工程体系中与正确性相关的软件规约,程序语义和程序验证等理论体系的变化,更进一步,多目标正确性概念还可能导致软件测试技术的新的发展。

习题 6.12

1. 什么是网构软件?
2. 网构软件有哪些特性?

参 考 文 献

[1] Alan Page Ken Johnston Bj Rollison. 微软的软件测试之道. 张爽,高博,欧琼,赵勇,等译. 北京:清华大学出版社,2009.

[2] Andreas Spillner, Tilo Linz, Hans Schaefer. Software Testing Foundations: A study Guide for the Certified Tester Exam 2nd Edition. 北京:人民邮电出版社,2008.

[3] 蔡建平. 软件测试大学教程(高等学校计算机应用规划教材). 北京:清华大学出版社,2009.

[4] 陈宝雷,徐丽,金海东. 软件测试工具实用教程. 北京:清华大学出版社,2009.

[5] 陈能技. 软件测试技术大全:测试基础、流行工具、项目实践. 北京:人民邮电出版社,2008.

[6] 陈汶滨、朱小梅、任冬梅. 软件测试技术基础. 北京:清华大学出版社,2008.

[7] 曹向志,于涌,高楼. 软件测试项目实战——技术、流程与管理. 北京:电子工业出版社,2010.

[8] Paul C Jorgensen. 软件测试. 韩柯,等译.北京:机械工业出版社,2003.

[9] 董杰. 软件测试精要. 北京:电子工业出版社,2009.

[10] 范勇. 软件测试技术. 西安:西安电子科技大学出版社,2009.

[11] Gerald M Weinberg. 完美软件——对软件测试的各种幻想. 宋锐,译. 北京:电子工业出版社,2009.

[12] 宫云战. 软件测试教程. 北京:清华大学出版社,2008.

[13] 古乐,史九林. 软件测试技术概论/软件测试系列. 北京:清华大学出版社,2004.

[14] 古乐,史九林. 软件测试案例与实践教程——高等学校教材软件工程. 北京:清华大学出版社,2007.

[15] K Mustafa, R A Khan. 软件测试:概念与实践. 董威,译. 北京:科学出版社,2009.

[16] 卡尼尔. 计算机软件测试. 王峰,译. 北京:机械工业出版社,2004.

[17] Louise Tamres. 软件测试入门. 包晓露,等译. 北京:人民邮电出版社,2004.

[18] 路晓丽,等. 软件测试技术. 北京:机械工业出版社,2007.

[19] 秦晓. 软件测试(新世纪计算机及相关专业系列教材). 北京:科学出版社,2008.

[20] 曲朝阳,等. 软件测试技术/21 世纪高等院校计算机系列教材. 北京:水利水电出版社,2006.

[21] Tom Amold Dominic Hopton. Visual Studio 2005 Team System. 软件测试专家教程. 颜炯,译. 北京:清华大学出版社,2008.

[22] 佟伟光. 软件测试. 北京:人民邮电出版社,2008.

[23] 王轶辰,等. 软件测试从入门到精通. 北京:电子工业出版社,2010.

[24] 魏伟. 笑傲测试——软件测试流程方法与实施. 北京:清华大学出版社,2006.

[25] William E. Perry. 软件测试的有效方法(第 3 版). 高猛,冯飞,徐璐,译. 北京:清华大学出版社,2008.

[26] 武剑洁,陈传波,肖来元. 软件测试技术基础. 武汉:华中科技大学出版社,2008.

[27] 郁莲. 软件测试方法与实践. 北京:清华大学出版社,2008.

[28] 赵斌. 高级软件测试工程师专用——软件测试技术经典教程. 北京:科学出版社,2007.

[29] 周伟明. 软件测试实践. 北京:电子工业出版社,2008.

[30] 周元哲. 软件测试教程. 北京:机械工业出版社,2010.

[31] 朱少民. 软件测试. 北京:人民邮电出版社,2009.

[32] 朱少民. 全程软件测试. 北京:电子工业出版社,2007.

[33] 朱少民. 软件测试方法和技术——高等学校教程·软件工程. 北京:清华大学出版社,2005.

[34] 梅尔斯(Myers G J). 软件测试的艺术中文版/(美). 王峰,陈杰,译. 北京:机械工业出版社,2006.
[35] 康一梅,等. 嵌入式软件测试. 北京:机械工业出版社,2008.
[36] 蔡建平. 嵌入式软件测试实用技术. 北京:清华大学出版社,2009.
[37] (美)斯顿(Sutton M)等. 模糊测试——强制性安全漏洞发掘. 黄陇,于莉莉,李虎,译. 北京:机械工业出版社,2009.
[38] 朱鸿,金凌紫. 软件质量保障与测试. 北京:科学出版社,1997.
[39] Dustin E,等. 自动化软件测试实施指南. 余昭辉,等译. 北京:机械工业出版社,2010.
[40] James A Whittaker. 探索式软件测试. 方敏,等译. 北京:清华大学出版社,2010.
[41] Paul Ammann,Jeff Offutt. 软件测试基础(英文版). 北京:机械工业出版社,2009.
[42] Aditya P Mathur. 软件测试基础教程(英文版). 北京:机械工业出版社,2008.
[43] Rick D Craig,等. 系统的软件测试. 杨海燕,等译. 北京:电子工业出版社,2003.
[44] 麦格雷戈,等. 面向对象的软件测试. 杨文宏,等译. 北京:机械工业出版社,2002.
[45] Rex Black. 软件测试实践——成为一个高效能的测试专家. 郭耀,等译. 北京:清华大学出版社,2008.
[46] Elfriede Dustin. 有效软件测试——提高测试水平的 50 条建议(影印版). 北京:中国电力出版社,2003.
[47] Elfriede Dustin 等. 软件自动化测试:引入,管理与实践. 于秀山,等译. 北京:电子工业出版社,2003.
[48] Seven R Rakitin. 软件验证与确认的最佳管理方法. 于秀山,等译. 北京:电子工业出版社,2002.
[49] 蔡为东. 步步为赢——软件测试管理全程实践. 北京:电子工业出版社,2009.
[50] 蔡为东. 赢在测试:中国软件测试先行者之道. 北京:电子工业出版社,2010.
[51] 魏伟. 笑傲测试——软件测试流程方法与实施. 北京:清华大学出版社,2006.
[52] 于涌. 精通软件性能测试与 LoadRunner 实战. 北京:人民邮电出版社,2010.
[53] 曹向志等. 软件测试项目实战:技术、流程与管理. 北京:电子工业出版社,2010.
[54] 梁博等. 测试有道:微软测试技术心得. 北京:电子工业出版社,2009.
[55] 王顺等. 软件测试方法与技术实践指南 ASP. NET 版. 北京:清华大学出版社,2010.
[56] James D McCaffrey. . Net 软件测试自动化之道. 刘晓伟,译. 北京:电子工业出版社,2007.
[57] 袁玉宇. 软件测试与质量保证. 北京:北京邮电大学出版社,2008.
[58] Changhai Nie,Hareton Leung. A survey of combinatorial testing. ACM Computing Survey,2011 43(2):1-29.
[59] Debra J Rechardson et al. Approaches to specification based testing,Proceedings of the ACM SIGSOFT,Third Symposium on Software Testing,Analysis,and Verification. ACM Press,1989:86-96.
[60] 颜炯,王戟,陈火旺. 基于模型的软件测试综述. 计算机科学,2004,31(2):184-187.
[61] L J Morell. A Theory of Fault-Based Testing. IEEE Transactions on Software Engineering,16(8):August 1990:844-857.
[62] P McMinn. Search-based software test data generation:A survey. Software Testing. Verification and Reliability,June 2004,14(2):105-156.
[63] Simon Poulding,John A Clark. Efficient Software Verification:Statistical Testing Using Automated Search. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,36(6):NOVEMBER/DECEMBER 2010:763-777.
[64] Yue Jia,Mark Harman. An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering,September 2011,37(5):649-678.
[65] G Fink,M Bishop. Property-Based Testing;A New Approach to Testing for Assurance. ACM SIGSOFT on Software Engineering,July 1997,22(4):74-80.

[66] Kai-Yuan Cai, Chang-Hai Jiang, Hai Hu, Cheng-Gang Bai. An experimental study of adaptive testing for software reliability assessment. The Journal of Systems and Software, 2008, 81: 1406-1429.
[67] 蔡开元等. 软件自适应测试方法. 中国专利，申请号:200410009711.1，公开日：2006.05.10.
[68] 崔展齐，王林章，李宣东. 一种目标制导的混合执行测试方法，计算机学报，2011，34(6)：953-964.
[69] Sen K, Marinov D, Agha G. CUTE: A concolic unit testing engine for C, Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13). New York, NY, USA: ACM, 2005: 263-272.
[70] Atif M Memon, Mary Lou Soffa, Martha E. Pollack. Coverage criteria for GUI testing. In ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering. New York: NY, USA, 2001: 256-267.
[71] Atif M Memon. GUI Testing: Pitfalls and Process Computer. IEEE Computer Society Press, 2002, 35(8): 87-88.
[72] DURAN J W, NTAFOS S C. An evaluation of random testing. IEEE Trans. Software Eng. 1984: 438-444.
[73] CHEN T Y, LEUNG H, MAK I. Adaptive random testing. In Proceedings of the Ninth Asian Computing Science Conference (ASIAN04). Springer Berlin/Heidelberg, 2004: 320-329.
[74] Y K Malaiya. Antirandom testing: getting the most out of black-box testing, Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering (ISSRE'95), Toulouse, France, October 1995:86-95.
[75] 刘攀，缪淮扣，曾红卫，等. 基于 FSM 的测试理论、方法及评估. 计算机学报，2011，34(6)：965-984.
[76] 聂长海. 关于软件测试的几点思考. 计算机科学，2011，38(2)：1-3，27.
[77] 聂长海，等. 配置测试的测试方案设计方法研究. 软件学报，2003，14(10)：146-154.
[78] Natarajan Meghanathan et al. BOOLEAN SPECIFICATION BASED TESTING TECHNIQUES: A SURVEY (Eds): ITCS, SIP, JSE-2012. Computer Science & Information Technology (CS & IT) 2012, 04:337-346.
[79] 邢学智. 基于 TTCN-3 语言的测试理论与技术研究. 合肥：中国科学技术大学，2010.
[80] 林惠民，张文辉. 模型检测：理论、方法与应用. 电子学报，2002，30(12A)：1907-1912.
[81] Hong Zhu, Xudong He. A methodology on testing high-level Petri nets. Information and Software Technology, 2002, 44: 473-489.
[82] Rodrigo M L et al. Inspections on Testing Aspect-Oriented Programs. 4th. Doctoral Symposium on Informatics Engineering, Porto, 6, February 2009.
[83] Reza Meimandi Parizi, Abdul Azim Abdul Ghani, Rusli Abdullah, Rodziah Atan. On the Applicability of Random Testing for Aspect-Oriented Programs. International Journal of Software Engineering and its Applications, October 2009, 3(4).
[84] 毛澄映，卢炎生. 构件软件测试技术研究进展. 计算机研究与发展，2006，43(8)：1375-1382.
[85] Muhammad Jaffar-ur Rehman et al. Testing software components for integration: a survey of issues and techniques. SOFTWARE TESTING, VERIFICATION AND RELIABILITY Softw. Test. Verif. Reliab. 2007: 17: 95-133.
[86] M A S Brito et al. Concurrent software testing: A systematic review. Technical Report 359, ICMC/USP, 2010.
[87] Jerry Gao, Xiaoying Bai, Wei-Tek Tsai. Cloud Testing-Issues, Challenges, Needs and Practice. SOFTWARE ENGINEERING: AN INTERNATIONAL JOURNAL (SEIJ), 1(1), SEPTEMBER 2011: 9-23.

[88] 陈火旺，王戟，董威. 高可信软件工程技术. 电子学报，2003，31(Z1)：2-7.
[89] 刘克，单志广，王戟等. 可信软件基础研究重大研究计划综述. 中国科学基金，2008，22(3)：145-151.
[90] 吕建，马晓星，陶先平等. 网构软件的研究与进展. 中国科学 E 辑，2006，36 (10)：1037-1080.
[91] 杨芙清，梅宏. Internet 时代的软件技术. 中国科学院技术科学论坛第十、十一次学术报告会论文集. 北京，中国科学院，2004.
[92] British Computer Society Specialist Interest Group in Software Testing(BCS SIGIST)，Standard for Software Component Testing，http://www.testingstandards.co.uk/